



I N T E R W O V E N

TeamSite® Templating: VisualFormat Developer's Guide

Release 5.5.1

© 2001-2002 Interwoven, Inc. All rights reserved.

No part of this publication (hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Interwoven. Information in this manual is furnished under license by Interwoven, Inc. and may only be used in accordance with the terms of the license agreement. If this software or documentation directs you to copy materials, you must first have permission from the copyright owner of the materials to avoid violating the law which could result in damages or other remedies.

Interwoven, TeamSite, OpenDeploy and the Interwoven logo are trademarks of Interwoven, Inc., which may be registered in certain jurisdictions.

SmartContext, DataDeploy, Content Express, the tagline and service mark are trademarks of Interwoven, Inc. which may be registered in certain jurisdictions. All other trademarks are owned by their respective owners.

TeamSite Templating utilizes third party components under the following copyrights with all rights reserved: Copyright 1999, Apache Software Foundation (www.apache.org); Copyright 1997-2000, Samizdat Productions; Copyright 1998-2001, Erik Bosrup (www.bosrup.com/web/overlib/). If you are interested in using these components for other purposes, contact the appropriate vendor.



Interwoven, Inc.
803 Eleventh Ave.
Sunnyvale, CA 94089
<http://www.interwoven.com>
Printed in the United States of America
Version 5.5.1
Part # 40-00-10-12-00-551-212

Table of Contents

Introduction	11
Defining the Toolbar.....	12
Modifying XML Configuration Data	12
Menus	13
Defining the VisualFormat Toolbar.....	13
Determining Which Menus Appear on the Toolbar.....	14
Determining Which Buttons and Selection Lists Appear on a Menu	17
Using JavaScript to Send Commands	24
Method Used to Send Commands Programmatically	24
Parameter Requirements for Commands.....	24
Creating a Custom Command	25
Defining a Custom Function for All Occurrences of the Editor.....	25
Writing a Custom JavaScript Event Function for One or More	
Occurrences of the Editor	28
Detecting When a Standard Command is Executed	29
Creating a Popup Menu.....	31
Determining which Fonts, Font Sizes, and Headings are Available	33
Changing Available Fonts	34
Changing Available Font Sizes	34
Changing Available Headings	34
Creating a List Item that Generates No Command.....	35
Dynamically Changing the Editor.....	37
Dynamically Creating Configuration Data on the Server Side	37
Avoiding Problems When Dynamically Changing the Toolbar on the	
Server.....	38
Dynamically Changing the Editor on the Client Using JavaScript	38
Disabling and Enabling Menu Items within Scripting	39
Accessing Menus and Commands.....	39
Enabling and Disabling a Command.....	40
Customizing Context Menus.....	41
Removing Commands from a Context Menu.....	41
Context Menu Commands and their Internal Names	42
Suppressing the Context Menu	42
Customizing the Popup Button	43
Customizing the createButton Command	44
Replacing the Popup Button with a Hyperlinked Image Button	44
Replacing the Popup Button with a Hyperlink	46

The Menus Object Interface.....	47
Defining Menus and Commands	47
Menu Object Quick Reference.....	48
Command Object Quick Reference	48
Script Example.....	50
Command Values	50
etbToolbarOptions	51
etbToolbarStyles	51
etbCaptionAlignment	51
etbToolbarLocation	52
etbToolbarModifications	52
etbCommandOptions	53
etbCommandStyles	53
etbCommandModifications.....	54
etbErrorValues.....	54
Menus Interface Definition	55
Method: SeparatorBarAdd.....	58
Method: SeparatorSpaceAdd	59
Command Item Interface	62
Modifying the Language of VisualFormat	70
Localization Files	70
Selecting a Default Language.....	71
Modifying ewebeditprodefaults.js	72
Modifying ewebeditpromessages.js	72
Selecting a Language Dynamically.....	72
Spell Checking in a Foreign Language.....	74
Customizable JavaScript Files.....	75
The ewebeditpro File	75
The ewebeditprodefaults File.....	76
The ewebeditpromessages File.....	80
Disabling the "Click OK to Preserve Changes" Message.....	82
The ewebeditproevents File.....	83
Client Installation Pages	84
Customizing the Client Installation Pages	84
Disabling the Installation Pages.....	85
JavaScript Objects.....	86
The VisualFormat Object	86
Properties	87
Methods	90
Events	93
Event Handler Functions	97

Reacting to the Creation of a Toolbar	99
Script Implementing a Toolbarreset Command	99
Script Reacting to a Toolbarreset Command	99
When the Command is Sent to the Script.....	100
Command Parameters	101
Implementing toolbarreset as a Toolbar Button.....	101
Using Toolbarreset to Reset Customization	101
The Redisplay Toolbars Command.....	102
The Instance Object.....	102
Properties	102
Methods	104
Events	105
The Parameters Object.....	106
Properties	106
ActiveX Control	115
Accessing the ActiveX Control.....	115
ActiveX Properties	115
ActiveX Methods.....	121
ActiveX Style Sheet Methods	124
ActiveX Events.....	131
The XML Configuration Data	132
Modifying XML Configuration Data	132
Organization of Configuration Documentation	133
Managing the Configuration Data	134
Editing the Configuration Data.....	134
Providing Configuration Files for User Groups	135
Changing the Configuration Data's Location	136
Allowing Users to Customize the VisualFormat Toolbar	137
Allowing User Customization	138
Preventing User Customization	139
Overriding User Customization.....	139
How VisualFormat Determines Which Configuration Data to Use.....	140
Overview of XML Configuration Data.....	141
Elements that Determine the User Interface	144
Hierarchy of Elements	145
The Config Element	146
The Interface Element.....	146
The Features Element.....	146
Attribute Types.....	147
Boolean	147
Integer	147
String	147

Element Definitions.....	148
bar	148
button	149
caption.....	150
command	152
cmd	154
config.....	155
features	156
image	157
interface	159
listchoice	161
mediafiles	164
menu	164
popup	166
selections	166
space.....	167
standard	168
toolTipText.....	169
 Commands.....	171
Topics Covered in This Section	171
Command Guidelines	172
Standard Commands	172
The cmdfontcolorvalue Command	175
Special Character Commands	177
Custom Commands	178
 Button Images.....	179
Formats Supported	179
Sources of Images.....	179
Images Supplied by VisualFormat	180
Creating Your Own Images.....	184
Image File Extensions	184
Size of Button Images	184
Background Color of Button Images	184
Button Image Specification Summary	185
 Table Commands	186
 Fonts and Headers	187
fonts	187
fontname.....	187
fontsize	189
headings	190
heading[x]	191
 External Features.....	193
Adding External Features	193
Examples	194

Viewing and Editing HTML Content	195
The ViewAs Feature	195
The EditHTML Feature	196
Cleaning HTML	197
Providing User Access to the Clean Feature.....	198
Clean Feature	198
Remove Element	200
Remove Endtag Element	200
Example	201
Remove Attribute Element Feature	201
Example	202
Removing Tags and Content Between Them	202
Example	203
The Spellcheck Feature	204
Spellcheck	204
Spellayt	205
Spellingsuggestion.....	205
Encoding Special Characters.....	207
Factors that Affect the Display of Special Characters.....	207
Displaying Asian Languages	209
Unicode Characters	209
Configuring VisualFormat for Extended and Special Characters.....	209
charencode Attribute	210
Choosing a charencode Value	213
References	215
Implementing a Web Site that Uses UTF-8 Encoding	216
Background Information.....	216
Implementing UTF-8	216
Setting the charset Parameter	217
Browser Support for UTF-8	218
For More Information about UTF-8	218
Style Sheets	219
Using Style Sheets to Standardize Formatting	220
The Default Style Sheet.....	220
Applying Style Sheets	220
Specifying a Style Sheet in the XML Configuration Data	221
Adding a Style Sheet to a Single Page	222
Dynamically Changing a Style Sheet for a Single Instance of the Editor	222
The BodyStyle Parameter.....	223
Preserving Tags When Office Content is Pasted.....	223
Saving Style Sheet Tags When Content is Saved	223
Setting Publishstyles to True.....	224
Setting Publishstyles to False	224

Implementing Style Class Selectors	224
Types of Style Classes	225
Translating Style Classes to a Foreign Language.....	226
Suppressing Styles from the Drop-down List	227
Managing Hyperlinks	228
Customizing Dropdown Menus in the Hyperlink Dialog Box.....	229
Customizing the Lists of the Hyperlink Dialog Box.....	229
Type List.....	230
Target Frame List	231
Quick Link List.....	232
Managing Image Selection	234
How Image Selection Works.....	234
Organization of the Image Selection Documentation	235
Examples of Implementing Image Selection.....	236
Example 1: No Restrictions, No Saving to a Database	236
Example 2: File Size Restriction, No Saving to Database.....	240
Example 3: FTP	244
Minimum Configuration Requirements for FTP	245
Server Configuration	245
Restriction Settings	246
User Interface Control.....	246
FTP Configuration in XML.....	247
Example 4: Database Samples	247
Implementing Image Upload	250
FTP File Upload.....	250
Security with FTP	251
HTTP File Upload	251
Overview	251
ASP	252
ColdFusion	254
Other Web Servers.....	255
The Mediafiles Feature of the XML Configuration Data.....	256
mediafiles.....	256
validext.....	256
maxsizek.....	257
transport.....	258
username.....	259
password	259
proxyserver.....	260
domain.....	260
xferdir.....	261
webroot.....	262

defsource	263
port.....	264
resolvemethod	264
Image Selection Object Methods and Properties.....	266
Image Selection Object Methods.....	266
Image Selection Object Properties	266
Programmatically Accessing MediaFile Properties	271
Accessing the Media File Object	271
Using Netscape to Access Image Properties	272
Entry Point for Using External Scripts.....	273
Setting External Page Parameters	274
Changing the Transfer Method on the Fly	275
Specifying an Image to Insert	275
Modifying the Upload Directory.....	276
Setting up an Image Repository.....	278
The Image Repository Folder	278
Inserting an Image into a Web Page	279
Example	280
Dynamically Selecting Upload Destinations.....	281
Summary	281
Implementing Image Upload.....	282
Background.....	282
MediaFile Object	282
Modifying the Upload Location.....	283
HTML Page	284
User Selection – Changing the Upload Location	285
Full Example	286
Integrating VisualFormat Using JavaScript.....	288
Formats for Placing the Editor on the Page	288
Creating Your Own Page	289
Create an HTML Page with Header and Body Tags	289
Include the VisualFormat JavaScript File	289
Enter a Form Element	289
Changing Parameter Values	290
Inserting the Editor as a Box	290
Declaring a Content Field after Creating the Editor.....	290
Inserting the Editor as a Button	292
Encoding Characters in the Value Attribute.....	293
Loading the Content	295
Detecting the Load Method	295
Manually Loading Content into the Editor	296
Saving the Content	296
Detecting when the Save Method is Invoked	297
Terminating the Save Method	297
Saving Content Manually	297

Closing a Window without Saving Content	297
Prevent Detecting the onsubmit Event.....	297
Prevent Detecting the onbeforeunload/onunload Event.....	298
Saving from One Instance of the Editor	298
Detecting When the Popup Editor is Activated	299
Appendix A: Naming the VisualFormat Editor	300
Appendix B: Error Messages	301

Introduction

This manual provides web developers with information they need to deploy and customize VisualFormat. It explains how to perform common tasks, such as removing a button from the toolbar and creating a custom command.

The manual also describes how to work with the following files.

NOTE

Typically, you would not change the java script files. Instead, you would create a new file, define a set of functions, and include this file in the HTML. You could also define the functions directly in the HTML file.

- JavaScript files
- the JavaScript objects
- the ActiveX control
- the locale files
- the configuration file
- ewebeditpropopup.htm

After you install VisualFormat, these files reside in your `iw/ewebeditpro20` directory. However, the configuration file is `/iw/config/visualformatconfig.xml`.

Finally, the manual explains the image upload feature and how to integrate VisualFormat onto a web page using JavaScript.

Defining the Toolbar

When you look at VisualFormat, you see a box with one or more rows of buttons across the top, known as the toolbar. The following illustrates a section of the VisualFormat toolbar.



When you first load VisualFormat, you see the default toolbar. You can determine which items appear on the toolbar, and what happens when a user selects an item by modifying the XML configuration data.

Modifying XML Configuration Data

There are two ways that you can modify XML configuration data

- *dynamically*, using JavaScript on the server, on the client, or both.
- *statically*, by editing an .xml file that stores the configuration data. The file's name isvisualformatconfig.xml, and by default, it is installed in the `iw/config` directory.

NOTE

If you use an XML editor to edit `visualformatconfig.xml`, a corresponding `.dtd` file (`config.dtd`) is supplied that you can use to validate `visualformatconfig.xml`. By default, the `config.dtd` is installed to the `iw/ewebeditpro20` directory.

This section explains how you can modify the toolbar and provides instructions for how to enact changes by editing the `visualformatconfig.xml` file. “Dynamically Changing the Editor” on page 37 explains how to change the toolbar using a script.

Menus

The toolbar includes one or more menus. Each menu consists of one or more toolbar buttons or *selection lists*. (In the above illustration, the word **Normal** in the lower left corner indicates a selection list.)

Here are key points about menus.

- A menu can reside on the same row with another menu. It can also continue to the next row if it cannot fit on a single row.
- Double vertical bars indicate the beginning of a new menu, as illustrated .
- You *can* place all available items on a single menu. However, it's probably more efficient to create a few menus that provide a related set of functions, and activate those menus in the configuration data assigned to a user group.

Defining the VisualFormat Toolbar

There are two major aspects to defining the VisualFormat toolbar. You can define

- which menus appear on the toolbar, and the sequence in which they appear
- characteristics of each button and selection list

You can also create a popup menu that appears when the user presses a button. Finally, you can create custom commands as well as add JavaScript that executes after a standard command is performed.

The following sections explain these procedures.

WARNING!

If you change the interface section of the XML configuration data, the user will not see the change if he or she has customized. For testing, to ensure that your changes appear, set the `allowCustomize` attribute of the `interface` element to "**false**" or change the `name` attribute of the `interface` element to a name not previously used. See also "Allowing Users to Customize the VisualFormat Toolbar" on page 137.

Determining Which Menus Appear on the Toolbar

When defining toolbar menus, you can perform the following tasks.

- Remove a menu
- Remove all toolbars
- Create a menu
- Determine whether a menu can reside on a row with another menu, or must appear on its own row
- If a menu does not fit on one row, determine if it should wrap around to the next row
- Create or edit the menu caption

Each task is described below.

Removing a Standard Menu

To remove a menu from the VisualFormat toolbar, follow these steps.

NOTE “Overview of XML Configuration Data” on page 141 explains how to edit the XML configuration file.

1. Within the interface section of the XML configuration data, locate the menu that you want to remove. (If you are unsure of the menu’s internal name, drag the menu from the toolbar. When you do, its caption appears. You can then search the XML configuration data for a menu with that caption.)



2. Within the definition of the menu you want to remove, set the enabled attribute to false. Here is an example. (The text appears in red for illustration purposes only.)

```
<menu name="editbar" enabled="false" newRow="false"
      showButtonsCaptions="false"
      wrap="false"> <caption localeRef="btnMainCap">Edit</caption>
        <button command="cmdcut"/>
        <button command="cmdcopy"/>
        <button command="cmdpaste"/>
        <button command="cmdfind"/>
        <bar/>
    </menu>
```

Users will not see the menu the next time they sign on.

Removing All Toolbars

To remove all toolbars from the VisualFormat editor, follow this step.

1. Within the interface section of the XML configuration data, set the `visible` attribute to **false**.

If the attribute does not appear, add it. Here is an example.

```
<interface name="standard" allowCustomize="false" visible ="true">
```

For more information, see “visible” on page 160.

Creating a New Toolbar Menu

Although you can type in a new menu definition, it is quicker to copy and edit one.

1. Within the interface section of the XML configuration data, copy and paste any menu definition (the text between the menu tags, `<menu` and `</menu>`).
2. Edit the menu’s name and other attributes as appropriate. (Menu attributes are explained in “menu” on page 164.)
3. Remove buttons that should not be available (see “Removing a Toolbar Button” on page 18).
4. Add new buttons as desired (see “Adding a Standard Toolbar Button” on page 17).

Users will see the new menu the next time they sign on.

Placing a Menu on a Row with Another Menu

A menu’s `newrow` attribute determines whether or not it can reside on the same row with another menu.

If the attribute is set to “**false**”, the menu resides on the same row



with another menu.

If "true", a menu goes to the beginning of the next row.



The default value for this attribute is "true".

To change the `newrow` attribute, follow these steps.

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
 2. Change the value of the `newrow` attribute.

Users will see the new menu arrangement the next time they sign on.

Determining if a Menu Should Wrap to the Next Row

A menu's `wrap` attribute determines what happens when a menu's toolbar buttons extend beyond the right edge of the menu row.

If the attribute is set to "true", when the icons reach the right edge of the display area, they wrap to the next row.

If "false", the icons do not wrap to the next row. They are invisible until you move the menu bar to another row or drag it from the toolbar.

The default value for this attribute is "true"

To change the `wrap` attribute, follow these steps

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
 2. Change the value of the `wrap` attribute.

Users will see the new menu arrangement the next time they sign on.

Creating or Editing the Menu Caption

A menu caption only appears when the user drags the menu away



In this example, "Edit" is the menu caption.

```
<menu  
<caption localeRef="mnueditcap">Edit</caption>  
</menu>
```

To change the menu caption, follow these steps.

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
2. Change the value of the `caption` text.

Users will see the new menu caption the next time they sign on.

Determining Which Buttons and Selection Lists Appear on a Menu

While the above section, "Determining Which Menus Appear on the Toolbar" on page 14, described how to modify menus, this section explains how to define the items that make up each menu. Menus are made up of toolbar buttons and selection lists. You can define the contents of menus in the following ways.

- Add a new toolbar button/selection list.
- Remove a toolbar button/selection list.
- Rearrange the toolbar buttons/selection lists on a menu.
- Add a space between two toolbar buttons/selection lists.
- Add a separator bar between two toolbar buttons/selection lists.
- Change the image that appears on a toolbar button.
- Display or suppress button caption text.
 - If you display caption text, you can define the alignment of the text on the button.
- Translate button captions and tool tips to a foreign language.

Adding a Standard Toolbar Button

As explained in "button" on page 149, buttons or selection lists execute commands. Standard and custom commands are defined in the features section of the XML configuration data. (See "Commands" on page 171 for more details.)

To add a new button or selection list to a menu, follow these steps.

NOTE

Whether the button appears as a square with an icon or a selection list is determined in the definition of the command, not when you create the button.

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
2. Move to the line where you want the new item to appear and enter the syntax to identify the new item. (The syntax for the button element is described in “button” on page 149.)

For example, if a menu definition already has three buttons, **cut**, **copy** and **paste**, and you want to add **find** following **paste**, move to the line following **paste** and add the **find** button command, as illustrated below (red indicates text that you insert).

```
<menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="true">
<caption localeRef="btnMainCap">Edit</caption>
    <button command="cmdcut"/>
    <button command="cmdcopy"/>
    <button command="cmdpaste"/>
    <button command="cmdfind"/>
</menu>
```

A list of standard commands is provided in “Standard Commands” on page 172.

Users will see the new menu arrangement the next time they sign on.

Removing a Toolbar Button

NOTE

This is the only element for which you cannot use the enabled property to have the editor ignore its values.

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
2. Move to the item that you want to remove.
3. To permanently remove the button, select the entire line and press <Delete>.

To temporarily remove the button, surround it with the characters that your XML editor uses to “comment out” text that is not executable code.

Users will see the new menu arrangement the next time they sign on.

Rearranging the Toolbar Buttons on a Menu

Buttons and selection lists appear on a menu in the sequence in which they are entered into the XML configuration data. For example, the following menu definition would create this menu



```
<menu name="editbar" .....>
<button command="cmdcut" />
<button command="cmdcopy" />
<button command="cmdpaste" />
<button command="cmdfind" />
<bar/>
</menu>
```

NOTE

The above illustration shows default images assigned to the commands in the example menu definition. However, you can modify these images using the image attribute of the command element.

To rearrange the toolbar buttons on a menu, follow these steps.

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
2. Move to the item that you want to move.
3. Select the entire line and cut it.
4. Move to the line where you want the item to appear and paste the text you cut in Step 3.

Users will see the new menu arrangement the next time they sign on.

Adding a Space Between Two Toolbar Buttons

You can add a space command to separate two menu items.



Buttons without a space command



Buttons with a space command

(For details, see “space” on page 167.)

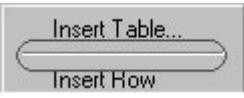
To add a space command, follow these steps.

-
1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
 2. Move to the item after which you want to insert the space and enter <space/>.

Users will see the new menu arrangement the next time they sign on.

Adding a Separator Bar Between Two Toolbar Buttons

Use the bar command to place a

- vertical bar  on a toolbar or
- horizontal bar  on a popup menu

To add a bar, follow these steps.

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
2. Move to the item after which you want to insert the space and enter <bar/>.

Users will see the new menu arrangement the next time they sign on.

Changing the Image that Appears on a Toolbar Button

The `image` element of the `command` element lets you specify the image that appears on a button. Each standard command has a default image.

To modify the image that appears on a button, follow these steps.

1. Within the interface section of the XML configuration data, move to the definition of the menu that you want to modify.
2. Move to the button whose image you want to change.
3. Identify the command assigned to the button. For example, in the following example, the command assigned to the first button on the menu named editbar is cmdcut.

```
<menu name="editbar" .....>
<button command="cmdcut"/>
```

```
<button command="cmdcopy" />
<button command="cmdpaste" />
<button command="cmdfind" />
<bar/>
</menu>
```

4. Move to the features section of the XML configuration data.
5. Find the command that you identified in Step 5.

```
<command name="cmdcut" enabled="true">
    <image key="Cut"/>
    <caption localeRef="btnTxtCut">Cut</caption>
    <toolTipText localeRef="btnTxtCut">Cut</toolTipText>
</command>
```

6. Replace the image element of the command with the name of the new image.

A list of standard images appears in “Images Supplied by VisualFormat” on page 180. As explained in “Sources of Images” on page 179, the `image` element can also refer to non-standard images.

Users will see the new image the next time they sign on.

Displaying Button Caption Text

Caption text appears on a toolbar button in the user interface.



NOTE

To determine the alignment of text within a button, edit the `textAlignment` attribute of the `menu` element. See “Defining the Alignment of Caption Text” on page 22.

To display caption text on a button, follow these steps.

1. Within the interface section of the XML configuration data, move to the definition of the command whose caption text you want to display.
2. Enable the `visible` attribute of the `command` element definition. (The default value of this attribute is “`false`”.) For example:

```
<command name="cmdcut" style="icon" visible="true">
    <image key="Cut"/>
    <caption localeRef="btnTxtCut">Cut</caption>
```

3. Move to the interface section of the XML configuration data.

-
4. Set the `showButtonsCaptions` attribute of the `menu` element to "true".

```
<menu name="editbar" showButtonsCaptions="true" ...
```

Users will see the caption text the next time they sign on.

To remove the display of caption text for a menu, reverse Step 6 above.

Defining the Alignment of Caption Text

You can set the alignment of button caption text using the `textalignment` attribute of the `menu` element. The possible alignment choices are listed below.

- top
- left
- right
- bottom
- center



You should only apply an `alignment` attribute to a button that displays caption text. By default, buttons do not display caption text. The procedure for displaying button caption text is described in "Displaying Button Caption Text" on page 21.

To change the alignment of button caption text, follow these steps.

1. Within the interface section of the XML configuration data, move to the definition of the menu whose button text alignment you want to change.
2. Edit a value for the `textalignment` attribute of the `menu` element definition. (The default value is "top".)

Possible values are:

- top
- left
- right
- bottom
- center

Users will see the new alignment of the caption text the next time they sign on.

Translating Button Captions and Tool Tips to a Foreign Language

As explained in “Modifying the Language of VisualFormat” on page 70, you can translate the language of VisualFormat’s user interface into several foreign languages. You can translate these elements of the interface.

- button caption text (if being displayed)
- tooltip text
- items on a pulldown menu
- items on a selection list

For any command in the XML configuration data, you use the `localeRef` attribute to assign a code that maps to a translation value in the localization file. For example, for the cut command, the standard `localeRef` value is `btnTxtCut`.

If a localization file is assigned to the `this.locale` element in the `ewebeditprodefaults.js` file, then VisualFormat performs these actions before displaying the toolbar.

1. Finds the code assigned to a command’s `localeRef` attribute.
2. Finds the corresponding value for the code in the localization file.
3. Displays the value from the localization file in the interface.

For example, if you assigned the German localization file in the `ewebeditprodefaults.js` file, and you are displaying button caption text, when VisualFormat launches, it

1. Finds the `localeRef` value for the cut button, `btnTxtCut`.
2. Finds the translation for that text in the German localization file, `<btnTxtCut>Ausschneiden</btnTxtCut>`.
3. Displays the text from the German localization file on the cut button, in this case, `Ausschneiden`.

To change the German text that appears, edit the localization file, in this case, `locale0407.xml`.

Using JavaScript to Send Commands

When adding custom buttons to the VisualFormat toolbar, you are typically interested in responding to a user button click. You may, however, wish to simulate the pressing of toolbar buttons using JavaScript. Every command defined in the configuration XML data can be executed from JavaScript as well as from user input.

The VisualFormat control exposes a method that can be called from JavaScript that causes the editor to perform the specified operation. The valid commands are listed in the configuration XML data.

Method Used to Send Commands Programmatically

Use this method to send these commands to the editor programmatically.

```
ExecCommand(strCmdName, strTextData, lData)
```

You can access this method through the ActiveX control (that is, in JavaScript) as either:

```
eWebEditPro["EditorName"].ExecCommand(...)
```

or

```
eWebEditPro.instances["EditorName" or index].editor.ExecCommand(...)
```

Examples:

```
eWebEditPro[sEditorName].ExecCommand("cmdcopy", "", 0);
eWebEditPro[sEditorName].ExecCommand("cmdfontcolor", strColorValue, 0);
```

Parameter Requirements for Commands

Most commands do not require parameters. For example, `cmdbold` simply bolds (or unbolds) the selected text, ignoring the `strTextData` and `IData` parameters.

For a complete list of commands and whether data parameters are used, see "Standard Commands" on page 172.

Creating a Custom Command

If you want to create a custom command for users to execute within the editor, you have two choices. You can

- define a custom JavaScript function
- write a custom JavaScript event function

Each method is explained below.

Defining a Custom Function for All Occurrences of the Editor

To define a custom JavaScript function, follow these steps.

1. Open a file (see “Where to Write the Custom JavaScript Function” on page 29).
2. Implement the following function. The function will be called when the user presses a button.

```
function eWebEditProExecCommand(sEditorName, strCmdName, strTextData, lData)  
{return true or false}
```

NOTE

If you are using Netscape 6 for editing, the parameters must be all lower case.

The following table explains the function’s arguments.

Argument	Description
sEditorName	The name of the occurrence of eWebEditPro whose command executes. To access the eWebEditPro methods, use eWebEditPro[sEditorName] . See Also: “Appendix A: Naming the VisualFormat Editor” on page 300
strCmdName	The name of the command that was just executed (if an internal command) or to be executed (if an external command).
strTextData	Any string data that the user passes to the command (typically not used).
lData	Any long integer value that the user passes to the command (typically not used).

-
3. Insert the JavaScript between the curly brackets ({}).
If the custom JavaScript returns **true**, the default external commands run. If it returns **false**, the commands do not run.
Internally handled commands, however, have already been executed prior to this event firing.
 4. Save the file.
 5. In the XML configuration data, assign the custom command to a toolbar button or menu. This procedure is documented in “Adding a Standard Toolbar Button” on page 17.
 6. In the XML configuration data, make the button or menu item available to users. This procedure is documented in “Adding a Standard Toolbar Button” on page 17.

Example of Creating a Custom Function

Assume that you want to create a custom button that, when pressed, inserts the company name. The company name is "My World Com" followed by the registered trademark symbol (®). The '®' value is the registered trademark symbol (®). The command name is "extcompany".

In the XML configuration data, add the command as a button on the "specialcharsbar" toolbar. In the features section of the XML configuration data, place the command in the external feature, since it is an external script feature.

XML File

```
<?xml version="1.0"?>
<config product="eWebEditPro">
    <interface name="standard" allowCustomize="false">

        .
        .

        <menu name="specialcharsbar" newRow="false" showButtonsCaptions="false" wrap="true">
            <caption localeRef="mnuSplChr" />

            .
            .

            <button command="extcompany" />
        </menu>

        .
        .

        <features>
            <external enabled="true">
                <command name="extcompany">
                    <image key="world"/>
                    <caption>Insert Company Name</caption>
                    <tooltiptext>Insert Company Name</tooltiptext>
                </command>
            </external>
        </features>
    </interface>
</config>
```

```
</command>

</config>
```

On the HTML page that displays the editor, insert the JavaScript function listed below. That function includes `eWebEditProExecCommand()`, a predefined command looked for in the Java core source code. When the user presses a toolbar button, `eWebEditProExecCommand()` is called to execute the command. In this example, the executed command is `extcompany`.

When the command is passed to the script, the code inserts the name of the company.

Note that this example assumes that your HTML page names the editor `MyContent1`. If the editor has a different name (check the line that begins with `eWebEditPro.create`), replace `MyContent1` with your editor name.

HTML File

```
<html>
<head>
.

<script language="JavaScript1.2">
//<!--
function eWebEditProExecCommand(sEditorName, strCmdName, strTextData, lData)
{
    if("extcompany" == strCmdName)
    {
        eWebEditPro.MyContent1.pasteHTML(" <HR>My World Com&#174;<BR> ");
    }

    return(true);
}
//-->
</script>

</body>
</html>
```

Preventing Default JavaScript From Executing

To prevent a default JavaScript function within `ewebeditproevents.js` from executing, add the following code to the bottom of the function (that is, to have the function return “`false`”).

```
return false;
```

Writing a Custom JavaScript Event Function for One or More Occurrences of the Editor

As an alternative to defining a custom function, you can write a custom JavaScript event handler function. To use this method of creating a custom command, follow these steps.

1. Write the custom JavaScript event handler (see “Where to Write the Custom JavaScript Function” on page 29).
2. Enter `eWebEditPro.onexeccommand = name_of_your_custom_event_handler` on the page where the editor is declared.
3. During the `onexeccommand` event, the following event object properties are available.

Event Object Property	Description
<code>eWebEditPro.event.srcName</code>	The name of the editor See Also: “Appendix A: Naming the VisualFormat Editor” on page 300
<code>eWebEditPro.event.cmdName</code>	The name of the command that the toolbar button executes
<code>eWebEditPro.event.textData</code>	Text entered by the user (typically not used)
<code>eWebEditPro.event.data</code>	Data entered by the user (typically not used)

4. In the XML configuration data, assign the custom event handler to a toolbar button or menu. This procedure is documented in “Commands” on page 171.
5. In the XML configuration data, make the button or menu item available to users. This procedure is documented in “Adding a Standard Toolbar Button” on page 17“.

To dynamically change the behavior of a command, change the value of `eWebEditPro.onexeccommand`.

Where to Write the Custom JavaScript Function

There are three places where you can write the custom JavaScript function.

Where	Sample Code	Use this approach to implement the function in
On the page with the editor See Also: "Appendix A: Naming the VisualFormat Editor" on page 300	<script language="JavaScript1.2"> enter JavaScript here eWebEditPro.create("MyContent1", 700,150); </script>	a single occurrence of the editor or all occurrences of the editor on that page
In a JavaScript file, referenced on each page that uses it	<script language="JavaScript1.2" src="yourfilename.js"> eWebEditPro.create("MyContent1", 700,150); </script>	all occurrences of the editor on that page
In a JavaScript file, referenced among the standard JavaScript files in the ewebeditpro.js file	var eWebEditProIncludes = ["ewebeditproevents.js", "ewebeditprodefaults.js", "your_custom_JavaScript_file.js" eWebEditProMsgsFilename, "ewep.js"];	all occurrences of the editor

Detecting When a Standard Command is Executed

You can use the eWebEditProExecCommand function to detect when a command has been executed. Then, within the function, you can add JavaScript to execute when the standard command executes.

For example, assume that whenever a user presses the underline toolbar button () , a custom warning message appears and tells the user that readers of the web page may mistake underlined words for hyperlinks.

Defining the Custom JavaScript Function

To define the custom JavaScript function, follow these steps.

1. Open a file (see "Where to Write the Custom JavaScript Function" on page 29.)

2. Implement the following function.

```
function eWebEditProExecCommand(sEditorName, strCmdName, strTextData, lData){}
```

The following table explains the function's arguments.

Argument	Description
sEditorName	The name of the editor in which the button was pressed. See Also: "Appendix A: Naming the VisualFormat Editor" on page 300
strCmdName	The name of the command that was just executed (if an internal command) or to be executed (if an external command).
strTextData	Any text that the user passes to the command (typically not used).
lData	Any data that the user passes to the command (typically not used).

3. Insert the JavaScript between the curly brackets ({ }).

Here is sample Javascript for this example.

```
function eWebeditProExecCommand (sEditorName,strCmdName,strTextData,ldata){  
    if (strCmdName=="cmdunderline"){  
        alert("Underlined text may be confused with hyperlinks");  
    }  
}
```

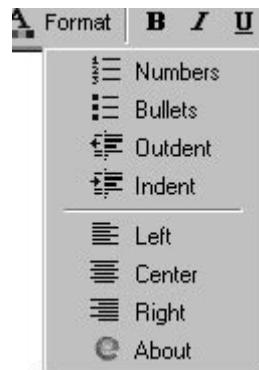
- If the command is standard (begins with cmd), the JavaScript executes after the standard command is executed.
- If the command is custom, the JavaScript executes before the command.

To continue with the above example, when the user presses the underline button, the editor applies underline. Next, the eWebEditProExecCommand function executes the JavaScript from within the command, which displays the warning message to the user.

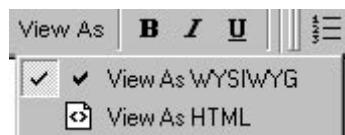
Creating a Popup Menu

You can assign a popup menu to a toolbar button. You might want to do this to provide access to many toolbar commands while limiting the toolbar to one row.

When a popup menu is assigned to a toolbar button, and a user clicks the button, the menu appears below the button, as illustrated below.



Follow these steps to create a popup menu. In this example, the menu appears on the toolbar with the text **View As**. When users click on the **View As** button, they see a menu with two choices, as illustrated below.



1. Within the features section of the XML configuration data, within the external feature, create a command to display the popup. Note that the command's caption will appear on the toolbar, indicating the menu's purpose. (For information about the syntax of the command element, see "command" on page 152.)

```
<command name="viewasselections" style="icon" visible="true">
    <caption localeRef="btnTxtVA">View As</caption>
    <toolTipText localeRef="btnVA">View As</toolTipText>
</command>
```

2. Move to the interface section of the XML configuration data and create a popup menu. Add to the menu the buttons that you want to make available.

In this example, you are adding one button for **ViewAsWYSIWIG** and another one for **ViewAsHTML**. (For information about the syntax of the `popup` element, see “[popup](#)” on page 166.)

```
<popup name="ViewAsPopup" localeRef="btnMyViewAs">View As</caption>
  <button command="cmdviewaswysiwyg" />
  <button command="cmdviewashtml" />
</popup>
```

3. Add to an existing or new menu a button that invokes the command you created in Step 1.

```
<menu name="ViewAsBar" newRow="false" showButtonsCaptions="false" style="false" >
  <caption localeRef="btnViewAs">View As...</caption>
  <button command="viewasselections" />
</menu>
```

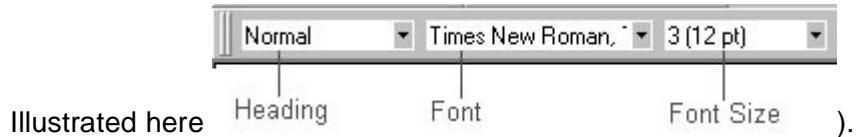
4. Within the button definition, specify a `popup` attribute. Enter the name of the menu you created in Step 3 as the value of the `popup` attribute.

```
<menu name="ViewAsBar" newRow="false" showButtonsCaptions="0" style="0" >
  <caption visible="0" localeRef="btnViewAs">View As...</caption>
  <button command="viewasselections" popup="ViewAsPopup" />
</menu>
```

Users will see the `popup` menu the next time they sign on.

Determining which Fonts, Font Sizes, and Headings are Available

When you install VisualFormat, the default toolbar provides the following fonts, font sizes and headings in selection boxes



Display item	Command name	Default Choices
Heading	cmdheaderlevel	<ul style="list-style-type: none">• Normal• Heading 1• Heading 2• Heading 3• Heading 4• Heading 5• Heading 6
Font	cmdfontname	<ul style="list-style-type: none">• Arial, Helvetica• Comic Sans MS• Courier New, Courier• Symbol• Times New Roman, Times• Verdana, Helvetica
Font size	cmdfontsize	<ul style="list-style-type: none">• 1 (8 pt)• 2 (10 pt)• 3 (12 pt)• 4 (14 pt)• 5 (18 pt)• 6 (24 pt)• 7 (36 pt)

Changing Available Fonts

To change the list of fonts available to users, follow these steps.

1. Within the features section of the XML configuration data, go to the line that begins `<command name="cmdfontname">`.
2. Delete or add fonts to the list.

NOTES

- A font only appears on a web page if it is installed on the computer of the user viewing the page. For this reason, we recommend only using standard HTML fonts.
- If more than one font is entered for a selection, the browser tries to display the first font. If it cannot find that font, it tries to use the second one, etc.

Users will see the new font list the next time they sign on.

Changing Available Font Sizes

To change the list of fonts available to users, follow these steps.
Note that the list cannot have more than seven sizes.

1. Within the features section of the XML configuration data, go to the line that begins `<command name="cmdfontsize">`.
2. Delete or add font sizes to the list. For example, to restrict users so that they can only apply font sizes 1, 3, 6 and 7, delete the lines for fonts 2, 4 and 5 (crossed out below).

```
<listchoice command="cmdfontsize1" localeref="mnuFS1">1 (8 pt)</listchoice>
<listchoice command="cmdfontsize2" localeref="mnuFS2">2 (10 pt)</listchoice>
<listchoice command="cmdfontsize3" localeref="mnuFS3">3 (12 pt)</listchoice>
<listchoice command="cmdfontsize4" localeref="mnuFS4">4 (14 pt)</listchoice>
<listchoice command="cmdfontsize5" localeref="mnuFS5">5 (18 pt)</listchoice>
<listchoice command="cmdfontsize6" localeref="mnuFS6">6 (24 pt)</listchoice>
<listchoice command="cmdfontsize7" localeref="mnuFS7">7 (36 pt)</listchoice>
```

Users will see the updated list of font sizes the next time they sign on.

Changing Available Headings

To change the list of headings available to users, follow these steps.

Note that the destination browser translates heading sizes into specific font sizes.

-
1. Within the features section of the XML configuration data, go to the line that begins <command name="cmdheaderlevel".
 2. Delete or add heading levels to the list.

For example, to restrict users so that they can only apply headings 1, 3 and 6, delete the lines for headings 2, 4 and 5 (crossed out below).

```
<selections name="headinglist" enabled="true" sorted="true">
    <listchoice command="cmdheadingstd" localeref="hdgtxtnorm">Normal</listchoice>
    <listchoice command="cmdheading1" localeref="hdgtxtlv11">Heading 1</listchoice>
    <listchoice command="cmdheading2" localeref="hdgtxtlv12">Heading 2</listchoice>
    <listchoice command="cmdheading3" localeref="hdgtxtlv13">Heading 3</listchoice>
    <listchoice command="cmdheading4" localeref="hdgtxtlv14">Heading 4</listchoice>
    <listchoice command="cmdheading5" localeref="hdgtxtlv15">Heading 5</listchoice>
    <listchoice command="cmdheading6" localeref="hdgtxtlv16">Heading 6</listchoice>
</selections>
```

Users will see the new heading list the next time they sign on.

Creating a List Item that Generates No Command

On the VisualFormat toolbar, you may want to create a drop down list whose first item describes the list, or is a default value. If the user selects the first item, no action should occur. Here is an illustration of such a drop down list.



You assign commands to list items in the XML configuration data. If a command is not assigned to an item, the command assigned to the list is sent. This section explains how to disable activity for a list selection.

Every command assigned to a list item is sent in the onexeccommand event. These commands do not need to be defined, as they do when listed on toolbars or popup menus.

Since the commands do not need to be defined, the values can be anything. If a command is defined that is not handled by the editor or by scripting, the command is ignored.

For example, you want to create a title for a list of colors, such as **Select Color**. To do this, assign a command that is not handled to the list item, for example, noop. You would define the list like this.

```
<selections name="fontcolorlist" enabled="true" sorted="true">
    <listchoice command="noop">Select Color</listchoice>
    <listchoice command="cmdfontcolor">Color palette</listchoice>
    <listchoice data="#FF0000">red</listchoice>
    <listchoice data="#0000FF">blue</listchoice>
    <listchoice data="#00FF00">green</listchoice>
</selections>
```

If a user selects **Select Color** from the drop down list, nothing happens.

Dynamically Changing the Editor

This section explains how to dynamically change the default options for each of VisualFormat's features using JavaScript

- on the server side, when the editor is created
- on the client side, after the editor is created

You can use these two approaches together if you wish.

Dynamically Creating Configuration Data on the Server Side

The server can dynamically create XML configuration data when the editor requests it. The data can be populated from a database. You can do this because the configuration of XML data is not limited to a flat file. This is one of XML's most powerful characteristics.

To use this method, set the editor's config parameter to a URL that returns configuration data as XML. The parameter must be set *before* the VisualFormat editor is created. Once the editor is created, the toolbar is static unless you also implement the technique described in "Dynamically Changing the Editor on the Client Using JavaScript" on page 38.

In the sample code below, the server (in this example, running ASP) returns XML data as it would appear in a flat visualformatconfig.xml file. But, the server is creating the file dynamically.

You use a URL parameter (**id=1**, in this example) to provide necessary information to the server. URL parameters are optional.

```
<script language="JavaScript1.2">
eWebEditPro.parameters.reset(); // set all parameters to their default values
// Request the configuration.
eWebEditPro.parameters.config = eWebEditPro.parameters.path + "config.asp?id=1";
</script>
```

Avoiding Problems When Dynamically Changing the Toolbar on the Server

If you plan to generate XML configuration data for the toolbar on the server, keep the following points in mind.

- If you are doing this with ColdFusion, you should turn off debug information. If you do not, you will receive a confusing error message.

You can turn off debug information site wide in the Cold Fusion Administrator Panel, under Debugging. Or, you can do this for a single page using the following code `<CFSETTING SHOWDEBUGOUTPUT="NO" >`.

- Caching may make it difficult to view differences in the editor. You should eliminate caching on the browser and also in the code.
For example, at the top of an ASP page, the following code forces the browser to flush the cache
`<RESPONSE.EXPIRES=0>`.
- If you dynamically create the toolbar in Netscape, the editor cannot access cookie information.

Dynamically Changing the Editor on the Client Using JavaScript

You can dynamically alter the toolbar after the editor is created using client-side JavaScript. You can add or remove buttons, drop-down lists, etc. Typically, the toolbar is changed in the editor's onready event as a result of user interaction.

Two sections of this manual provide the information about client-side scripting.

- “Dynamically Changing the Editor on the Client Using JavaScript” on page 38 provides tips and techniques for using client scripting to access menu functionality.

-
- “The Menus Object Interface” on page 47 provides the API definition of the Menus object contained within the VisualFormat interface. The Menus object interface contains properties and methods that let you control menu, button, and command functionality.

Disabling and Enabling Menu Items within Scripting

You can use client scripting to access menu functionality. This includes menu creation, command creation, and display status. This section describes how to disable and enable a menu item through scripting.

Accessing Menus and Commands

To access menus, use the Toolbars method of the VisualFormat control. This method returns a reference to the menu control object.

```
var objInstance = ewebEditPro.instances[seditorname];  
var objMenu = objInstance.editor.Toolbars();
```

To access a command, use the following method in the Menu object.

Method: CommandItem(CommandName As String) As CCommandItem

(See Also: “Method: CommandItem” on page 56)

The following are methods in the Command object used to affect enable status.

Method: setProperty(Name As String, Value As Variant)

See Also: “Method: setProperty” on page 69

Method: getPropertyBoolean(Name As String) As Boolean

See Also: “Method: getPropertyBoolean” on page 67

Property: CmdGray As Boolean

See Also: “Property: CmdGray As Boolean” on page 64

Enabling and Disabling a Command

To enable or disable a command, first retrieve the interface to menus using the Toolbar method:

```
var objInstance = eWebEditPro.instances[seditorname];
var objMenu = objInstance.editor.Toolbars();
```

Then, use the Menu object to gain access to a specific command.

```
var objCommand = objMenu.CommandItem("cmdcut");
```

To disable an icon command, set the CmdGray property to true.

```
objCommand.setProperty("CmdGray", true);
```

To enable an icon command, set the CmdGray property to false.

```
objCommand.setProperty("CmdGray", false);
```

Example

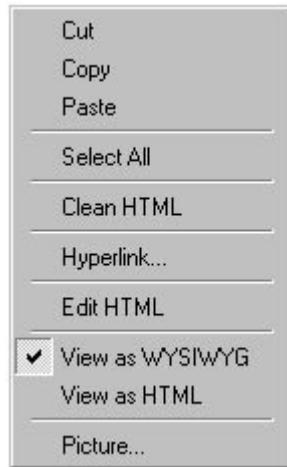
Below are functions that disable and enable a command item.

```
function DisableCommand(seditorname,scommandname)
{
  var objInstance = eWebEditPro.instances[seditorname];
  var objMenu = objInstance.editor.Toolbars();
  objMenu.CommandItem(scommandname).setProperty("CmdGray", true);
}

function EnableCommand(seditorname,scommandname)
{
  var objInstance = eWebEditPro.instances[seditorname];
  var objMenu = objInstance.editor.Toolbars();
  objMenu.CommandItem(scommandname).setProperty("CmdGray", false);
}
```

Customizing Context Menus

This section describes how to customize context-sensitive menus that appear when a user right-clicks the mouse. For example, the following menu appears if text has been selected when the user right-clicks the mouse within the editor.



A different context menu appears if no text was selected or if the cursor is within a table.

The section explains how to remove commands from the context menus, and how to suppress them. Currently, you cannot *add* items to a context menu.

Removing Commands from a Context Menu

To remove a command from the context menu, set its `enabled` attribute to “**false**”. Note that this change also removes the command from the toolbar menu. You cannot remove a command from the context menu and leave it on the toolbar menu.

For example, to remove the copy command from the context menu, edit the XML configuration data as illustrated below. (To

learn more about editing XML configuration data, see “Modifying XML Configuration Data” on page 132.)

```
<standard>
<command name="cmdCopy" enabled="false">
</command>
</standard>
```

Context Menu Commands and their Internal Names

In order to remove a command from a context menu, you must know its internal name. This table lists the internal name of most commands.

Menu Command	Internal Name
Cut	cmdcut
Copy	cmdcopy
Paste	cmdpaste
Select all	<i>cannot be removed</i>
Clean HTML	cmdclean
Hyperlink...	cmdhyperlink
Edit HTML	edithtml
Insert HTML	edithtml
View as WYSIWYG	cmdviewaswysiwyg
View as HTML	cmdviewashtml
Picture...	cmdmfumedia

Suppressing the Context Menu

The XML configuration data’s interface element has a `context` attribute that determines whether context menus appear when a user right clicks the mouse.

```
<config product="eWebEditPro">
<interface name="standard" visible ="true" allowCustomize="false" context="false">
```

If the `context` attribute is set to “**true**”, the context menu appears when the right mouse is clicked. If the attribute is set to “**false**,” the context menu is suppressed.

The default value for `context` is “**true**”.

Customizing the Popup Button

This section explains how to customize a popup button and window.

There are two approaches to customizing the popup button and window. You can

- Change values in `ewebeditprodefaults.js` and `ewebeditpromessages.js`. All properties within these files that start with `popup` affect the popup button or window. For example, the popup button caption is declared as `popupButtonCaption` in `ewebeditpromessages.js`.
- Change the properties using JavaScript. Two objects, `buttonTag` and `popup`, are part of the `parameters` object. For example,

```
eWebEditPro.parameters.buttonTag.value="Edit Description";
eWebEditPro.createButton("btnDesc", "Desc");
```

NOTE

You *cannot* use JavaScript at run time to change popup properties in `ewebeditpromessages.js` or `ewebeditprodefaults.js`.

The JavaScript objects correspond as shown below.

Parameter	<code>ewebeditprodefaults.js</code>
<code>buttonTag.start</code>	<code>popupButtonTagStart</code>
<code>buttonTag.value</code>	<code>popupButtonCaption</code> (in <code>ewebeditpromessages.js</code>)
<code>buttonTag.end</code>	<code>popupButtonTagEnd</code>
<code>popup.url</code> <i>See Also:</i> "popup" on page 111	<code>popupurl</code>
<code>popup.windowName</code>	<code>popupWindowName</code>
<code>popup.windowFeatures</code>	<code>popupWindowFeatures</code>
<code>popup.query</code>	<code>popupQuery</code>
<code>styleSheet</code> <i>See Also:</i> "Style Sheets" on page 219	<code>styleSheet</code>

"Customizable JavaScript Files" on page 75 explains how to edit the JavaScript files. The rest of this section explains how to customize the popup button using JavaScript.

Customizing the `createButton` Command

By default, when you create a popup edit button for VisualFormat (using the `<input type=button>` element), a standard HTML button with a caption of **Edit** is created.

In JavaScript, use the `createButton` method to create the button.

```
eWebEditPro.createButton("btnName", "contentFieldName");
```

To customize the button text with JavaScript, use a parameter object. For example

```
eWebEditPro.parameters.buttonTag.value = "Edit with eWebEditPro";
```

You can further customize the popup with a few lines of JavaScript. Three essential parameters (listed below) are used to create a popup button.

parameters.buttonTag	Defines	Default Value in File
start	The leading part of the HTML to create the button.	ewebeditprodefaults.js: popupButtonTagStart
value	The value attribute.	ewebeditpromessages.js: popupButtonCaption
end	The trailing part of the HTML to create the button.	ewebeditprodefaults.js: popupButtonTagEnd

Replacing the Popup Button with a Hyperlinked Image Button

To create a button that consists of a hyperlink tag and an image tag, set the start and end parameters to generate the proper HTML.

For example, assume that you want to enter the following HTML code.

```
<a href="..."><img alt='Your description' width=X height=Y src='yourbutton.gif' ...></a>
```

Since there is no value attribute, set the value parameter to an empty string. Here is an example of how to set the parameters (the content field name appears in red).

```
<script language="JavaScript">

eWebEditPro.parameters.buttonTag.start = "<a href='javascript:eWebEditPro.edit(\"MyContent1\")'><img alt>Edit width=150 height=60 src=button.gif onclick=''"';

eWebEditPro.parameters.buttonTag.value = "";

eWebEditPro.parameters.buttonTag.end = "></a>";

</script>
```

WARNING!

Note that onclick is set to an empty string (shown in green). If you omit this attribute, two edit windows open when using Internet Explorer or Netscape 6.

Add the code shown above just before you call the function to create the popup button. Since you must explicitly assign the editor name (in the example, the name is `MyContent1`), copy this code for each call to create a popup button.

For example, to create two buttons in JavaScript, you could use the following code (content field names appear in red).

```
<textarea name=Summary rows=5 cols=70>

<script language="JavaScript1.2">

eWebEditPro.parameters.reset();

eWebEditPro.parameters.buttonTag.start = "<a href='javascript:eWebEditPro.edit(\"Summary\")'><img alt>Edit width=150 height=60 src=button.gif onclick=''"';

eWebEditPro.parameters.buttonTag.value = "";

eWebEditPro.parameters.buttonTag.end = "></a>";

eWebEditPro.createButton("btn1", "Summary");

</script>

...

<textarea name=Description rows=10 cols=70>

<script language="JavaScript1.2">

eWebEditPro.parameters.reset();

eWebEditPro.parameters.buttonTag.start = "<a href='javascript:eWebEditPro.edit(\"Description\")'><img alt>Edit width=150 height=60 src=button.gif onclick=''"';
```

```
eWebEditPro.parameters.buttonTag.value = "";  
eWebEditPro.parameters.buttonTag.end = "></a>";  
eWebEditPro.createButton("btn2", "Description");  
</script>
```

Replacing the Popup Button with a Hyperlink

To replace the popup button with a hyperlink, add the following code just before you call the function to create the popup button.

Since you must explicitly assign the editor name (in the example, the name is `MyContent1`), copy this code for each call to create a popup button.

In this example, the content field name appears in red.

```
<script language="JavaScript">  
eWebEditPro.parameters.buttonTag.start = "<a href='javascript:eWebEditPro.edit(\"MyContent1\")' onclick=''"';  
eWebEditPro.parameters.buttonTag.value = "";  
eWebEditPro.parameters.buttonTag.end = ">edit</a>";  
</script>
```

For example, to create two buttons in JavaScript, you could use the following code (content field names appear in red).

```
<textarea name=Summary rows=5 cols=70>  
<script language="JavaScript1.2">  
eWebEditPro.parameters.reset();  
  
eWebEditPro.parameters.buttonTag.start = "<a href='javascript:eWebEditPro.edit(\"Summary\")' onclick=''"';  
eWebEditPro.parameters.buttonTag.value = "";  
eWebEditPro.parameters.buttonTag.end = ">Edit Summary</a>";  
eWebEditPro.createButton("btn1", "Summary");  
</script>  
...  
  
<textarea name=Description rows=10 cols=70>  
<script language="JavaScript1.2">  
eWebEditPro.parameters.reset();  
  
eWebEditPro.parameters.buttonTag.start = "<a href='javascript:eWebEditPro.edit(\"Description\")' onclick=''"';  
eWebEditPro.parameters.buttonTag.value = "";  
eWebEditPro.parameters.buttonTag.end = ">Edit Description</a>";  
eWebEditPro.createButton("btn2", "Description");  
</script>
```

The Menus Object Interface

This chapter describes the API definition of the Menus object contained within the VisualFormat interface. It assumes that you have moderate expertise in JavaScript, HTML, and ActiveX technology.

The Menus object interface contains properties and methods that let you control menu, button, and command functionality.

To retrieve the Menus object, use the VisualFormat Toolbar method.

```
var objMenu = objEditor.Toolbars();
```

Defining Menus and Commands

The menus and commands are typically defined in the visualformatconfig.xml file assigned to the editor. This file defines the command names, how they look, and where they appear in the user interface.

You can add scripting to modify or supplement the XML data. The server side can anticipate changes required for a user and dynamically generate a customized XML from database information. To accomplish this, it may be necessary to interact with commands at a level below user interaction. This low level interaction can include creating a toolbar using commands supported by a script and removing commands not supported by a script.

This chapter contains information to help a developer perform these operations. In doing so, it provides a reference to the API.

Menu Object Quick Reference

Below is a list of the methods available to the Menus object.

Command	For more information, see page
CommandAdd	55
CommandDelete	56
CommandItem	56
HideAbout	57
HideAllMenus	57
PopupMenu	57
SeparatorBarAdd	58
SeparatorSpaceAdd	59
ShowAbout	59
ShowAllMenus	60
ToolbarAdd	60
ToolbarModify	61

Command Object Quick Reference

Below is a list of the available methods and properties for the CommandItem object.

Method/Property	For more information, see page
Method: AddItem	62
Method: Clear	63
Property: CmdCaption As String	63
Property: CmdData As Long	63
Property: CmdGray As Boolean	64
Property: CmdIndex As Integer	64
Property: CmdName As String	64
Property: CmdStyle As Integer	64
Property: CmdText As String	64

Method/Property	For more information, see page
Property: CmdToggledOn As Boolean	64
Property: CmdToolTipText As String	64
Property: CmdType As etbCommandStyles	64
Property: CmdVisible As Boolean	65
Method: FirstCommand	65
Method: getProperty	65
Method: getPropertyBoolean	66
Method: getPropertyInteger	66
Method: getPropertyString	66
Method: isValid	67
Method: ListCommandName	68
Property: MaxListboxWidth As Integer	68
Method: NextCommand	68
Method: setProperty	69

Script Example

The following code is a sample JavaScript 1.2 function that creates a menu, creates buttons and commands, and grays out command items.

```
function AffectMenusAndCommands(seditorname)
{
    var objInstance = eWebEditPro.instances[seditorname];
    var objMenu = objInstance.editor.Toolbars();

    // This shows how to create a menu.
    objMenu.ToolbarAdd("SampleMenu", "This is an External Menu", 0, 0, 0, 0, "");

    // This shows how to add commands to a menu.
    objMenu.CommandAdd("jssample1", "Sample Command 1", "Sample Command 1", "key", 0, 0, "SampleMenu", 0,-1);
    objMenu.CommandAdd("jssample2", "Sample Command 2", "Sample Command 2", "key", 0, 0, "SampleMenu", 0,-1);
    objMenu.CommandAdd("jssample3", "Sample Command 3", "Sample Command 3", "key", 0, 0, "SampleMenu", 0,-1);
    objMenu.CommandAdd("jssample4", "Sample Command 4", "Sample Command 4", "key", 0, 0, "SampleMenu", 0,-1);
    objMenu.CommandAdd("jssample5", "Sample Command 5", "Sample Command 5", "key", 0, 0, "SampleMenu", 0,-1);

    // This shows how to disable an existing command.
    objMenu.CommandItem("cmdcut").setProperty("CmdGray", true);
    objMenu.CommandItem("cmdpaste").setProperty("CmdGray", true);
}
```

Command Values

The following command values are collections of API parameter value types.

- etbToolbarOptions
- etbToolbarStyles
- etbCaptionAlignment
- etbToolbarLocation
- etbToolbarModifications
- etbCommandOptions
- etbCommandStyles
- etbCommandModifications
- etbErrorValues

etbToolbarOptions

Menu creation options are bitwise or'ed together.

Value	Description
0x1	Invisible
0x2	New menu row
0x4	ShowCaption
0x8	Wrap rather than scroll
0x10	Show tool tips
0x20	Show customize selection menu button
0x40	Floating

etbToolbarStyles

Toolbar Styles. Used when creating a toolbar.

Value	Description
0	Icon Bar
1	Pulldown Menu
2	Tab List
3	Popup or context menu

etbCaptionAlignment

Caption alignment of buttons. Use these values when creating a toolbar or command.

The effects are seen only when a toolbar is floating or a command is defined to show its caption.

See Also: "Defining the Alignment of Caption Text" on page 22.

Value	Align caption to the
0	No alignment of caption
1	Top
2	Bottom
3	Left
4	Right
5	Center

etbToolbarLocation

Toolbar Location. Used when creating a toolbar.

Value	Align toolbar to the
0	No alignment of toolbar. Default (top) is used.
1	Top
2	Bottom
3	Left
4	Right
5	Form

etbToolbarModifications

Toolbar modifications allowed. Used when modifying a toolbar.

Value	Description
0	Delete command wherever it is.
1	Remove from toolbar but keep on customization list.
2	Move from the selection to the toolbar.
3	Set as pressed in or checked.
4	Set as un-pressed or unchecked.
5	Toolbars only - clear all commands.
6	Disable the item.
7	Enable the item.

etbCommandOptions

Command creation options are bitwise or'ed together. Used when creating or modifying a command.

Value	Description
0x1	Invisible
0x2	Initially shows as disabled

etbCommandStyles

Command Styles. Used when creating a command.

Value	Description
0	Use the Default button style when the function it represents has no dependence on other functions. For example, a Save File operation can be performed at any time. Further, when the button is depressed, it springs back again when the function is finished.
1	The Check style should be used when the function it represents is a toggle of some kind. For example, when using a RichTextBox control, selected text can be either bold or not. Thus, when the button is depressed, it stays depressed until it is pressed again.
2	Not supported.
3	The separator style has no function except to create a button that is eight pixels wide. Use the separator style to create a button that separates one button from another. See Also: "Adding a Separator Bar Between Two Toolbar Buttons" on page 20 Or, use it to enclose the group of buttons with the ButtonGroup style.
4	The placeholder style functions as a "dummy" button. Use this button to create a space on the Toolbar control where you want to have another control appear (such as a ComboBox or ListBox control).
5	Drop down list box.
6	Text edit box.

etbCommandModifications

Command modifications allowed. Used when modifying a command.

Value	Description
0	Delete all instances of a command.
1	Command is removed from the toolbar but kept on the customization list.
2	Display command on toolbar or menu.
3	Set as pressed in or checked.
4	Set as un-pressed or unchecked.
5	For toolbars only; clear all commands.
6	Disable the item.
7	Enable the item.

etbErrorValues

Error definitions. These are returned by the Menus interface methods.

Value	Description
0	No error.
1	Functionality not supported with this version.
2	No toolbars have been created for any operation.
3	Invalid ID given for a command or toolbar.
4	Unknown location requested for text or command.
5	Internal error.
6	The specified toolbar does not exist.
7	Error using definition file.
8	Definition not found.
9	Configuration can't be used, even if given.
10	Error creating item.

Menus Interface Definition

Method: CommandAdd

Command with Parameters

```
CommandAdd(CommandName As String, CommandCaption As  
String, ToolTip As String, ImageFileAs String, Options As Long,  
Style As etbCommandStyles, Optional ToolbarName As String  
= "", Optional MaxWidth As Long = 0, Optional iPosition As Integer =  
-1) As CCommandItem
```

Description

Adds a command to the specified toolbar.

Parameters

Parameter	Type	Description
CommandName	String	The name of the command to add. When selected, this is the string value sent up as the command. See Also: "Commands" on page 171
CommandCaption	String	The caption to use next to the command.
ToolTip	String	Tool tip text that pops up when the cursor hovers over a command.
ImageFile	String	The file to use as the icon image. This can also be one of the internal image definitions.
Options	Long	Bit field of etbToolbarOptions bits describing specific options for the toolbar. See Also: "etbToolbarOptions" on page 51.
Style	Long	The style from the etbCommandStyles set of values. See Also: "etbCommandStyles" on page 53.
ToolbarName	String	The name of the toolbar to attach this command to. If left blank, it is not assigned to a toolbar but is available for customization.

Return

This command returns a reference to the command item that was created. Be sure to check that the command is *not* nothing (that is, null) before using it.

Method: CommandDelete

Command with Parameters

CommandDelete(CommandName As String, Optional
ToolbarName As String = "", Optional iIndex As Integer = -1)

Description

This command deletes a command from a toolbar. If a toolbar name is not specified, it is deleted from all locations.

Parameters

Parameter	Type	Description
CommandName	String	The command to remove.
ToolbarName	String	The toolbar from which to remove the command. If this is blank, or not included, the command is removed from all toolbars.

Return

There is no return value from this routine.

Method: CommandItem

Command with Parameters

CommandItem(CommandName As String) As CCommandItem

Description

Retrieves the interface directly to the command item.

For a list of methods and properties available to the CommandItem object, see “Command Object Quick Reference” on page 48.

Parameters

Parameter	Type	Description
CommandName	String	The command to retrieve.

Return

This returns a reference directly to the command item.

Method HideAbout() As Boolean

Description

Hides the about command button, if it is shown.

NOTE

It is better to use the `ShowAbout` property, contained within the **VisualFormat** interface.

Parameters

Parameter	Type	Description
none		

Return

This returns the previous setting for hide.

Method HideAllMenus()

Description

Quickly hides all toolbar menus.

Parameters

Parameter	Type	Description
none		

Return

There is no return value with this item.

Method PopupMenu

Command with Parameters

```
Sub PopupMenu(MenuName As String, Optional RelativeCmd As String = "")
```

Description

Brings up a popup menu.

Parameters

Parameter	Type	Description
MenuName	String	The name of the Popup Menu to bring up. If the menu does not exist or is not a popup menu style, nothing happens
RelativeCmd	String	The command associated with the popup. Optional

Return

There is no return value.

Method: SeparatorBarAdd

Method with Parameters

Method: SeparatorBarAdd(CommandName As String, ToolbarName As String, Optional iPosition As Integer = -1) As Boolean

Description

Adds a separator bar to the specified toolbar. On a toolbar, it is a vertical bar. On a popup menu, it is a horizontal bar. It is mostly used to organize commands into groups.

See Also: "Adding a Separator Bar Between Two Toolbar Buttons" on page 20.

Parameters

Parameter	Type	Description
CommandName	String	The separator bar is assigned an internal name. This value receives that name. It is used as a reference if it is modified.
ToolbarName	String	The name of the toolbar or menu to which to add the bar.
iPosition	Integer	Position of the command within the given toolbar. If omitted or - 1, it is placed at the end.

Return

This returns true if it successfully created the separator.

Method: SeparatorSpaceAdd

Method with Parameters

SeparatorSpaceAdd(CommandName As String, ToolbarName As String, Optional iPosition As Integer = -1) As Boolean

Description

Adds a separator space to the specified toolbar. It is used mainly to organize commands into groups.

Parameters

Parameter	Type	Description
CommandName	String	The separator space is assigned an internal name. This value receives that name. It is used as a reference if it is modified.
ToolbarName	String	The name of the toolbar or menu to which to add the bar.
iPosition	Integer	Position of the command within the given toolbar. If omitted or -1, it is placed at the end.

Return

This returns true if it successfully created the separator.

Method ShowAbout() As Boolean

Description

Shows the about button if defined in the XML data.

NOTE

It is better to use the ShowAbout property, contained within the **VisualFormat** interface.

Parameters

Parameter	Type	Description
none		

Return

This returns the previous show state of the about button.

Method ShowAllMenus()

Description

Restores the view of menus hidden with HideAllMenus().

Parameters

Parameter	Type	Description
none		

Return

There is no return value.

Method ToolbarAdd

Method with Parameters

```
Function ToolbarAdd(ToolbarName As String, Caption As String,  
CaptionAlignment As etbCaptionAlignment, Style As  
etbToolBarStyles, Options As Long, Position As  
etbToolBarLocation, Optional ParentMenu As String = "") As  
Integer
```

Description

Creates a toolbar and adds it to the toolbars available to the user.

Parameters

Parameter	Type	Description
ToolbarName	String	The name of the toolbar. This must be unique among all currently created toolbars.
Caption	String	The toolbar caption.
CaptionAlignment	etbCaptionAlignment	The alignment of the toolbar caption. See Also: "etbCaptionAlignment" on page 51.
Style	etbToolbarStyles	The style of the toolbar. See Also: "etbToolbarStyles" on page 51.
Options	Long	Bit field of etbToolbarOptions bits describing specific options for the toolbar. See Also: "etbToolbarOptions" on page 51.
Position	etbToolbarLocation	Toolbar position. See Also: "etbToolbarLocation" on page 52
ParentMenu	String	The name of the parent menu. This is for use with sub-menus. Optional.

Return

Returns an etbErrorValues value.

Method ToolbarModify

Method with Parameters

Function ToolbarModify(ToolbarName As String, Modification As etbToolbarModifications) As Integer

Description

Modifies an existing toolbar.

Parameters

Parameter	Type	Description
ToolbarName	String	The name of the toolbar to change.
Modification	etbToolbarModification	How to modify the toolbar. See Also: "etbToolbarModifications" on page 52

Return

Returns an etbErrorValues value.

Command Item Interface

The following is a full list of available methods and properties.

Method: AddItem

Method with Parameters

AddItem(ItemText As String, Optional ItemData As Long = 0,
Optional strCmdName As String = "")

Description

In an edit control, this method sets the text. In a list box, it adds an item to the selection list. Otherwise, it does nothing.

Parameters

Parameter	Type	Description
ItemText	String	The text of the selection.
ItemData	Long	Data associated with the command. If this is omitted or 0 (zero), the data returned with the selection is the 0-based index into the list.
StrCmdName	String	Command to associate with the list selection. If this is a value, the specified command name is sent to the client in place of the command that contains the list.

Return

Nothing.

Method: Clear()

In a list box, this method clears all entries. In an edit box, it clears the text. In a toggle, it ensures that it is untoggled.

Parameters

Parameter	Type	Description
none		

Return

Nothing

Property: CmdCaption As String

Retrieves the caption. If a special button, the caption is a key word.

Property: CmdData As Long

If this is a list item, this property sets the current item to the entry that contains the long data value associated with the text. For a combo-box, it is either the long value given to the item or the index into the item. For a text box, it is the length of the string. For a toggle, it is the 1/0 (on/off) state.

Property: CmdGray As Boolean

If set to true, the command is disabled and displayed as a grayed image. The button does not produce a command when selected by the user. If set to true, the command is available to the user.

Property: CmdIndex As Integer

This property only applies to list items. It sets the currently selected index and retrieves the currently selected index into the list box.

Property: CmdName As String

This returns the command name associated with the button. If the command name of a list item is required, use
`ListCommandName().`

Property: CmdStyle As Integer

This property reflects the style of the command. The style is assigned when the command is created.

This is a read-only property.

Property: CmdText As String

This property sets the current selection for a list box. It sets the edit text for an edit box. The text is displayed on the button, no matter what.

Property: CmdToggledOn As Boolean

This property is only available to buttons that are created with the Toggle style. If the value is true, the button appears pressed in or selected. If false, it appears popped out or unselected.

Property: CmdToolTipText As String

This property contains the tooltiptext associated with a command. You can modify the tooltip through this property.

Property: CmdType As etbCommandStyles

The command type assigned during the creation of the command.

This is a read-only property.

Property: CmdVisible As Boolean

This property reflects the visibility of a command. If true, the user can see the command. If false, the command is invisible.

Do not use this property to disable buttons. Use the `CmdGray` property instead.

If the button is made invisible, an empty space replaces the button.

Method: FirstCommand

Method with Parameters

`FirstCommand(strName As String, strCaption As String) As Boolean`

Description

Sets the current reference to the first command available. The reference value held by the script does not change. The reference change is internal to the command mechanism.

To further any enumeration, see “Method: NextCommand” on page 68.

Parameters

Parameter	Type	Description
StrName	String	Receives the name of the first command.
StrCaption	String	Receives the caption of the command. If a text item, it is the text. If a list box, it is the currently selected item text.

Return

If true is returned, it was able to find a command.

Method: getProperty

Method with Parameters

`getProperty(Name As String) As Variant`

Description

Retrieves the property name given.

This method provides Netscape compatibility.

It is better to use the other getProperty methods to return the correct type. If this method is used, the data type is not guaranteed.

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return

The data as a variant. The data type is not guaranteed.

Method: getPropertyString

Method with Parameters

getPropertyString(Name As String) As String

Description

Retrieves the property name given as a string.

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return

The data of the property as a string.

Method: getPropertyInteger

Method with Parameters

getPropertyInteger(Name As String) As Long

Description

Retrieves the property name given as an integer.

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return

The data of the property as an integer.

Method: getPropertyBoolean

Method with Parameters

getPropertyBoolean(Name As String) As Boolean

Description

Retrieves the property name given as a boolean.

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return

The data of the property as a boolean.

Method: IsValid() As Boolean

Is the current command object a valid command? This should be checked after retrieving a command.

Parameters

Parameter	Type	Description
None		

Return

If this method returns true, the current command being referenced is valid.

Method: ListCommandName

Method with Parameters

ListCommandName(idx As Integer) As String

Description

Available only with list box commands. Returns the name of the command associated with the item at the index specified.

Parameters

Parameter	Type	Description
idx	Integer	The 0-based index into the list of commands.

Return

The command name associated with the index. If no command is associated, either the name of the list command or nothing is returned.

NOTE To retrieve the index of the selected list item, use the CommandItem's CmdIndex property: objCommand .CmdIndex; or
objCommand .getPropertiesInteger("CmdIndex").

Property: MaxListboxWidth As Integer

Sets or retrieves the width of an edit box or a list box in characters.

Method: NextCommand

Method with Parameters

NextCommand(strName As String, strCaption As String) As Boolean

Description

Sets the current reference to the next command available. The reference value held by the script does not change. The reference change is internal to the command mechanism.

To initiate any enumeration, see also “Method: FirstCommand” on page 65.

Parameters

Parameter	Type	Description
StrName	String	Receives the name of the first command.
StrCaption	String	Receives the caption of the command. If a text item, it is the text. If a list box, it is the currently selected item text.

Return

If true is returned, the method found a command. If it returns false, there are no more commands to enumerate. The reference will be on the last command enumerated.

Method: setProperty

Method with Arguments

setProperty(Name As String, Value As Variant)

Description

Sets the named property to the value given.

See the getProperty series of methods (beginning with "Method: getProperty" on page 65) on how to retrieve values.

Parameters

Parameter	Type	Description
Name	String	The name of the property.
Value	Variant	The data to set into the property.

Return

Nothing

Modifying the Language of VisualFormat

You can specify a foreign language for all instances of the editor, or you can set the language dynamically on a web page. You can also spell check your content in a foreign language.

See also: “Translating Button Captions and Tool Tips to a Foreign Language” on page 23.

Localization Files

Localization files are XML files that translate VisualFormat’s menus and dialog boxes from English to another language. VisualFormat provides locale files for several languages.

NOTE If you use an XML editor to edit locale.xml, a corresponding .dtd file (locale.dtd) is supplied that you can use to validate locale.xml. By default, the locale.dtd is installed to the `iw/ewebeditpro20` directory.

To translate to a language not provided by Ektron, see Ektron’s Language Translation Kit (available on <http://www.ektron.com/support.cfm>).

Each localization file name includes a four-character code that specifies the language and country. For example, `locale0807.xml` = German (Switzerland).

Within each localization file, an `xml:lang` attribute specifies the two-letter language and country code. For example, `xml:lang="de-ch"` for German (Switzerland).

Your system’s localization files reside by default in `mywebsite.com/iw/ewebeditpro20/` and begin with **locale**.

The following table lists the standard locale files that Ektron delivers with VisualFormat, and the language of each file. More files are available in Ektron's Language Translation Kit.

Locale code	Language
0000	English
0404	Chinese (simplified)
0406	Danish
0407	German
0409	English
040a	Spanish
040c	French
0410	Italian
0411	Japanese
0412	Korean
0413	Dutch
0816	Portuguese (standard)

See Ektron's web site for more information on language support.

Selecting a Default Language

To specify a default language for VisualFormat, modify these files.

File	Determines the language of
ewebeditprodefaults.js	menus and dialog boxes
ewebeditpromessages.js	system messages and status bar text
the spelling dictionary in the version of Microsoft Word installed on the client	the spell checker

Modifying ewebeditprodefaults.js

Within ewebeditprodefaults.js, set the Locale ActiveX control property to refer to the localization file for the language of your choice.

NOTE The Locale ActiveX control property may contain XML content, but it typically refers to a directory or a localization file.

To match the localization file to the regional settings of the client computer, set the Locale ActiveX control property to the VisualFormat directory. For example,

```
this.locale = this.path
```

To specify a particular language, set the Locale ActiveX control property to the URL of that language's localization file. For example,

```
this.locale = this.path + "locale0000.xml";
```

Modifying ewebeditpromessages.js

To translate system messages (including those with HTML tags) and the text on the status bar to the language of your choice, modify the ewebeditpromessages.js file. (For more information, see “The ewebeditpromessages File” on page 80.)

Selecting a Language Dynamically

To dynamically select a language for VisualFormat, follow these steps.

1. Specify a localization file. The dynamic web server should assign the file name.

For ColdFusion, set the Locale attribute to the URL of the locale file of your choice. For example,

```
<CF_eWebEditPro ... Locale="/iw/ewebeditpro20/locale0000.xml" ...>
```

For other platforms, add a SCRIPT element to assign the locale. On the web page, you must set the locale *before* you insert an editor. For example,

```
<script language="JavaScript1.2">
eWebEditPro.parameters.locale = "/iw/ewebeditpro20/locale0000.xml";
</script>
```

-
2. Specify the language-specific ewebeditpromessages.js file.
Copy and translate the ewebeditpromessages.js file for each language. The file name should be meaningful, such as, ewebeditpromsgsfrench.js.
 3. Edit the ewebeditpro.js file by replacing "ewebeditpromessages.js" with a JavaScript variable, for example, eWebEditProMsgsFilename.

So,

```
var eWebEditProIncludes =
"ewebeditproevents.js",
"ewebeditprodefaults.js",
"ewebeditpromessages.js",
"classmodel.js",
"axelement.js",
"eweppar.js",
"ewepfind.js",
"ewepedit.js",
"ewepalt.js",
"eweppopup.js",
"ewepfactory.js"];
```

becomes

```
var eWebEditProIncludes =
"ewebeditproevents.js",
"ewebeditprodefaults.js",
eWebEditProMsgsFilename,
"classmodel.js",
"axelement.js",
"eweppar.js",
"ewepfind.js",
"ewepedit.js",
"ewepalt.js",
"eweppopup.js",
"ewepfactory.js"];
```

4. Add a SCRIPT element dynamically to the HEAD section of your web page.

The SCRIPT element assigns the name of the language-specific ewebeditpromessages.js file to the eWebEditProMsgsFilename variable. Be sure to define the eWebEditProMsgsFilename variable in the web page *before* you include the ewebeditpro file.

NOTE

Because ColdFusion does not include the **eWebEditPro** file, you can place the SCRIPT element anywhere within the head.

For example,

```
<head>
  <script language="JavaScript1.2">
    var eWebEditProMsgsFilename = "ewebeditpromsgsfrench.js";
  </script>
  ...
</head>
```

Spell Checking in a Foreign Language

When you are spell checking the content inserted into VisualFormat, the spell checker refers to the spelling dictionary in Microsoft Word. If the dictionary in the client's version of Microsoft Word is not English, the VisualFormat's spell check feature will refer to that dictionary.

Customizable JavaScript Files

VisualFormat provides four JavaScript files that let you customize many attributes of the HTML element used to place the ActiveX control in a web page. The files let you customize how the editor appears in a web page.

The customizable files are

- ewebeditpro.js
- ewebeditprodefaults.js
- ewebeditpromessages.js
- ewebeditproevents.js

This section describes each file.

The ewebeditpro File

The ewebeditpro.js file has only two elements, described below.

Element	Description
LicenseKeys	Enter the license keys if they were not entered during installation, or change them as needed. For details on how to do this, see the “License Keys” chapter in the Installation/Integration manual for your platform.
eWebEditPro Path	Enter or edit the path to the directory to which VisualFormat is installed.

The ewebeditprodefaults File

The ewebeditprodefaults.js file has many attributes. The following table describes some of them. The rest of them are described in "ActiveX Control" on page 115.

As described in "Changing Parameter Values" on page 290, you would make changes to this file that apply to *all* occurrences of the editor. To change any of these values for a *single* occurrence of the editor, you would insert JavaScript onto the page that invokes the editor.

Default	Description
path	The directory path to which VisualFormat is installed. This value is set in the ewebeditpro.js file, and only displayed in this file.
clientInstall	The directory in which the client installation file (ewebeditproclient.exe) resides.
Popup Window Defaults	This set of defaults determines the attributes of the instance of VisualFormat that appears as a popup window.
popupURL	The URL to the web page that contains the editor that appears in the popup window. The default value is <code>this.path + "ewebeditpropopup.htm";</code>
popupWindowName	The name assigned to the popup window created by the standard JavaScript function <code>window.open()</code> .

Default	Description
popupWindowFeatures	<p>The parameters passed to the standard JavaScript <code>window.open()</code> method.</p> <p>To enable a feature (e.g., scrollbars), include the keyword. To disable a feature, exclude the keyword.</p> <p>Separate each feature keyword by a comma, but <i>include no</i> spaces between parameters.</p> <p>A few of the possible features include:</p> <ul style="list-style-type: none"> • <code>width=x</code>, where <code>x</code> is the window width in pixels • <code>height=y</code>, where <code>y</code> is the window height in pixels • <code>scrollbars</code>: displays scrollbars • <code>status</code>: displays the status bar • <code>resizable</code>: the user can change the window size • <code>location</code>: displays the location (or address) bar • <code>menubar</code>: displays the menu bar • <code>toolbar</code>: displays toolbar buttons <p>For more details on the JavaScript <code>window.open()</code> method, see a JavaScript reference.</p>

Default	Description
popupQuery	<p>Lets you pass query string parameters to the popup web page (specified in <code>popupUrl</code> property). By default, the <code>popupUrl</code> page is a static HTML page, but it could be a dynamically generated page. In either case, you may want to pass additional information to the popup page. For example, you may want to display the number of times the content has been edited, the title of the content, or anything else.</p> <p>The <code>popupQuery</code> property must not include the question mark (?) character.</p> <p>The JavaScript <code>escape()</code> function ensures the text is saved to pass in a URL. For example, it changes all space characters to %20. The <code>unescape()</code> function restores the text.</p> <p>Here is an example that passes a title and instructions relevant to the content being edited.</p> <p>On the page with the popup button:</p> <pre data-bbox="687 771 1291 1072"><script language="JavaScript"> var sTitle = "Summary Description"; var sInstr = "Please enter a paragraph summarizing the page."; eWebEditPro.parameters.popupQuery = "title=" + escape(sTitle) + "&instr=" + escape(sInstr); ... eWebEditPro.createButton(...); </script></pre> <p>On the popup page:</p> <pre data-bbox="687 1118 1305 1647">var objQuery = new Object(); var strQuery = location.search.substring(1); var aryQuery = strQuery.split("&"); var pair = []; for (var i = 0; i < aryQuery.length; i++) { pair = aryQuery[i].split("="); if (pair.length == 2) { objQuery[unescape(pair[0])] = unescape(pair[1]); } } document.writeln("<p>" + objQuery["title"] + "</p>"); document.writeln(objQuery["instr"] + "
");</pre>

Default	Description
popupButtonTagStart, popupButtonTagEnd	Defines the button that launches the popup window.
maxContentSize	The largest number of characters that can be saved in the editor window. If a user enters content that exceeds this size, an error message appears.
ondblclickelement	<p>The name of the JavaScript event that occurs when a user double-clicks on a hyperlink, applet, object, image or table within the editor. The default event handlers are defined in the ewebeditproevents.js file.</p> <p>See also "ActiveX Events" on page 131.</p>
embedAttributes	Optional attributes to the EMBED tag. Applies only to Netscape.
objectAttributes	<p>Optional attributes to the OBJECT tag. Applies only to Internet Explorer.</p> <p>For example, the OBJECT tag has an attribute 'standby' that the developer could set to a string.</p> <pre>objectAttributes="standby='Please wait while the editor initializes.'";</pre>
editorGetMethod	<p>Lets you retrieve either the contents of the body or the entire HTML document from the editor.</p> <p>The possible values are "getBodyHTML" or "getDocument". They are documented in "ActiveX Control" on page 115.</p>

The following attributes are documented in "ActiveX Control" on page 115.

- license
- locale
- config
- hideAboutButton

The onexeccommand attribute is described in "The ewebeditproevents File" on page 83.

The ewebeditpromessages File

The attributes in the ewebeditpromessages.js file determine the text of buttons and popup messages that appear when using VisualFormat.

Attribute	Determines the text that appears	Default Message
popupButtonCaption	On the button that launches the popup window that contains VisualFormat.	Edit
installPrompt	In a dialog box when the client installation is needed.	Click OK to install VisualFormat
waitingToLoad	In the status bar while the editor is waiting to load.	Waiting to load
loading	In the status bar while the editor is loading.	Loading
doneLoading	In the status bar when the editor finishes loading.	Done loading
errorLoading	In the status bar when the editor encounters an error and cannot load.	Error loading
saving	In the status bar when the editor is saving content.	Saving
doneSaving	In the status bar when the editor has saved content.	Done saving

Attribute	Determines the text that appears	Default Message
querySave (used only with Internet Explorer)	<p>In a dialog box if the user moves to another page before the content is saved. (Note: The content is saved when the user clicks the submit button.)</p>	<p>Click OK to preserve changes when moving to another page. Click Cancel to discard changes.</p> <p>NOTE</p> <hr/> <p>This message only appears when using Internet Explorer.</p> <hr/> <p>See also "Disabling the "Click OK to Preserve Changes" Message" on page 82.</p>
confirmAway (used only with Internet Explorer)	<p>In a dialog box if the user indicates that he/she does not want to save changes.</p>	<p>Any changes will be lost.</p>
saveFailed	<p>In a dialog box if the user clicked save but the editor cannot save the content.</p>	<p>Unable to save. Continue and lose content?</p>
sizeExceeded	<p>In a dialog box if the amount of content that user wants to save exceeds the maxContentSize.</p>	<p>Content is too large to save. Please reduce the size and try again.</p>
clientInstallMessage	<p>If the user opens a page with the editor in Netscape when VisualFormat is not yet installed. The message appears below the textarea element that appears in place of the editor.</p>	<pre>
 VisualFormat is not installed. Click to install VisualFormat .</pre>

Attribute	Determines the text that appears	Default Message
elementNotFoundMessage	<p>When VisualFormat cannot find the named content element.</p> <p>The message appears below the editor.</p>	' Unable to find content field (typically a hidden field) within a form. Please check the following: Form tag is required Content field is required and must match the name specified when creating the editor Content field must be declared prior to creating the editor Name specified: '
invalidFormMethodMessage	<p>When the form element's method is not set to post.</p>	' The form method must be set to "post". For example, <form method="post">; The submit will fail using "get".'

Disabling the "Click OK to Preserve Changes" Message

If a user attempts to move to another web page, the **Click OK to preserve changes when moving to another page. Click Cancel to discard changes** message appears.

If you want to suppress the message, add the following JavaScript to a page with the editor.

```
eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_SAVE or EWEP_ONUNLOAD_NOSAVE
```

WARNING!

If you disable this message, a user's edits will be lost unless the user pressed the Submit button before moving to another page.

The ewebeditproevents File

When the user presses a button on the toolbar or double-clicks an element (for example, a hyperlink or image) in the content, an event is raised. When the event fires, it can cause a JavaScript function to run.

The ewebeditproevents.js file contains JavaScript event handler functions that perform actions. These actions could include inserting HTML into the content (for example, the trademark symbol) and opening a window to a hyperlink that was double-clicked.

You can edit ewebeditproevents.js to modify, remove, or add to the JavaScript functions. In this way, you can customize what happens when the event fires.

Event	Determines How to Respond When
onDoubleClickElementHandler	A user double-clicks.
onDoubleClickHyperlinkHandler	A user double-clicks on a hyperlink.
onExecCommandHandler	A toolbar button is pressed or the user selects an item from the context-sensitive menu.

Client Installation Pages

The first time a user on a client PC accesses a web page using Internet Explorer that has a new or upgraded version of VisualFormat, an .htm page appears. The page provides information about the installation and prompts the user to continue with the installation or cancel.

By default, this .htm page is named intro.htm, and is installed in /iw/ewebeditpro20/clientinstall/intro.htm.

Note: Previous versions of VisualFormat should be uninstalled prior to installing a new version.

Another .htm page, installnow.htm (installed into the same directory) is invoked from intro.htm. installnow.htm displays **Please wait while VisualFormat is being installed**, then notifies the user whether or not the installation was successful.

Customizing the Client Installation Pages

If you want to customize these .htm pages (for example, to change the text), save the file under a different name and make changes to the copy. Otherwise, future upgrades will overwrite your changes.

If you change the intro.htm file, you also need to change the reference to the file and pathway in the ewebeditprodefaults.js file. In that file, the intro.html page is defined at the this.installPopupUrl property, as illustrated below.

```
function eWebEditProDefaults()
{
.
.
.
this.installPopupUrl = this.path + "clientinstall/intro.htm";
```

If you want to display the intro.htm page before loading a page that includes VisualFormat, you may call the

`eWebEditPro.autoInstallExpected()` method to determine if the client computer would automatically install VisualFormat.

To popup a window with the intro.htm page in JavaScript, call

```
eWebEditPro.installPopup.open();
```

Disabling the Installation Pages

If you want to disable the client installation pages, you have two choices.

- In `ewebeditprodefaults.js`, set `this.installPopupUrl = " "`;
- In JavaScript, set `eWebEditPro.parameters.installPopup = null;`

JavaScript Objects

The VisualFormat Object

This chapter describes the VisualFormat JavaScript object's properties, methods, and events. It also describes the Instance object and the Parameters object.

Properties

Property	Value	Description
{editor name}		A reference to an instance of the VisualFormat ActiveX control. See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.
actionOnUnload (applicable to IE only)	EWEP_ONUNLOAD_SAVE: (default)	When the web page is unloaded, the content is saved to a hidden field on the web page without prompting. The content is posted to the server only when the user clicks a Submit button.
	EWEP_ONUNLOAD_NOSAVE:	When the web page is unloaded, the content is <i>not</i> saved to a hidden field. WARNING: <i>All changes</i> since the last save are lost. For example, if the user presses the Back button, content in standard HTML elements is preserved, but any changes made in the VisualFormat editor are lost.
	EWEP_ONUNLOAD_PROMPT:	When the web page is unloaded (except for submit), a dialog box prompts the user to click OK to save changes. (The dialog box text is defined by querySave in ewebeditpromessages.js). If the user clicks Cancel , another dialog box prompts whether to discard the changes or stay on the same web page. (The dialog box text is defined by confirmAway in ewebeditpromessages.js).
instances collection		An array of in-line editor objects of type eWebEditProEditor or eWebEditProAlt (if the editor could not be created). This array may be indexed by an integer (0 to instances.length-1) or by the name of an instance of an editor (for example, instances ["Editor1"]). The eWebEditProEditor object has an editor property that provides a reference to the VisualFormat ActiveX control. For more information, see "Event Handler Functions" on page 97.
installPopup	true or false	If true, a window with the intro.htm page pops up. See also: "Client Installation Pages" on page 84.

Property	Value	Description
isAutoInstallSupported	true or false	If true : VisualFormat can be automatically installed. Currently, automatic installation is only supported on IE 5.0 or later. If false : VisualFormat cannot be automatically installed. The client installation program is required to install VisualFormat on the client computer.
isInstalled	true or false	If true : VisualFormat is installed (or presumed installed). For Netscape, this indicates the Esker plug-in was installed. For IE, this indicates the editor is installed. If false : VisualFormat is supported in this environment but needs to be installed on the client. By default, a message will appear prompting the user to install the client software.
isSupported	true or false	If true : VisualFormat is supported in this environment. It may not be installed yet. If false : VisualFormat cannot run in this environment.
parametersobject		An object of type eWebEditProParameters containing the default set of parameters used when creating an instance of the editor or button. To edit the default values set for the parameters, edit the ewebeditprodefaults.js file. For more information, see "The Parameters Object" on page 106.
status	<ul style="list-style-type: none"> • EWEP_STATUS_INSTALLED • EWEP_STATUS_NOTLOADED • EWEP_STATUS_LOADING • EWEP_STATUS_LOADED • EWEP_STATUS_SAVING • EWEP_STATUS_SAVED • EWEP_STATUS_NOTSUPPORTED • EWEP_STATUS_NOTINSTALLED • EWEP_STATUS_FATALERROR • EWEP_STATUS_UNABLETOSAVE • EWEP_STATUS_SIZEEXCEEDED 	Reflects the current state of VisualFormat.

Property	Value	Description
upgradeNeeded	true or false	<p>If true: An older version VisualFormat is installed and needs to be upgraded. Currently only available with IE.</p> <p>If false: VisualFormat is either the same or newer version, or could not be determined.</p>
versionInstalled		<p>This property retrieves the version of the control. It is a comma delimited list with this format:</p> <p>Major Major, Minor Major, Major Minor, Minor Minor</p> <p>Or</p> <p>Version, Point Release, just 0, Revision (The Major Minor value is not used, so it is always 0.)</p> <p>Syntax</p> <pre>strData = [form!]eWebEditPro2.Version</pre> <p>Remarks</p> <p>The Major Minor value of 0 is in the format because of the agreed upon format for software module versions. If comparing versions, the string must be parsed and each item converted to an integer.</p> <p>Examples</p> <p>Displays the control version:</p> <pre>function ShowVersion() { alert(testItem1.Version); } or alert(eWebEditPro.MyContent1.version); or alert(eWebEditPro.MyContent1.getPropertyString("version"));</pre> <p>Currently only available with IE</p>

Methods

Method	Description	Arguments
autoInstallExpected	<p>This method returns a boolean that indicates whether an automatic download and installation of VisualFormat is expected or not. This value can be used to display a message informing the user what to expect while VisualFormat is installed.</p> <p>See Also: "Client Installation Pages" on page 84</p>	<p>true: automatic installation is supported and VisualFormat is either not installed or requires upgrading.</p> <p>false: either automatic installation is not supported or the correct version of VisualFormat is installed.</p>
create (name, width, height, parameters)	<p>Creates an instance of an in-line editor in the page. Returns an instance object, which is also added to the instances array. If successful, the editor's name is added to the VisualFormat object to permit easy access to the ActiveX control.</p>	<p>name - Name of the editor. Must match the name of a standard HTML element (typically an input type= hidden) unless the content is to be manually loaded and saved. If the editor is placed on the popup editor page (e.g., ewebeditpropopup.htm), the name is arbitrary.</p> <p>See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.</p> <p>width - The width of the editor in pixels or a percent. For example, 700 or "100%".</p> <p>height - The height of the editor in pixels. If the editor cannot be displayed because VisualFormat is not supported or not installed, a textarea element appears in its place as close in size as possible. Textarea size is specified in rows and columns instead of pixel width and height. You can specify the rows and columns in the parameters object.</p> <p>parameters (optional) - Optional parameters object for VisualFormat. If not specified, the parameters in the VisualFormat object are used. Parameters supplied to the popup editor take precedence over these.</p>

Method	Description	Arguments
createButton(buttonName, elementName, parameters)	<p>Creates an instance of a button which, if clicked, opens a popup window with the editor in it.</p> <p>This method must be called even when a custom button is used instead of the standard HTML button. The creation of a standard HTML button may be suppressed by clearing the parameters.buttonTag, in which case, the custom button must call the edit method.</p>	<p>buttonName - The name of the HTML button element to create. The caption that appears on the button is defined as popupButtonCaption in ewebeditpromessages.js. (See "The ewebeditpromessages File" on page 80.)</p> <p>elementName - The name of the HTML element that stores the content. The element name may contain the form name to differentiate elements of the same name in different forms, for example: "frmMain.Content".</p> <p>This name is passed to the edit method when the button is clicked.</p> <p>parameters (optional) - Optional parameters object for VisualFormat. If not specified, the parameters in the VisualFormat object is used.</p>
edit(elementName) EstimateContentSize (ContentType As String) As Long	<p>Opens a popup window with the editor in it. This method is called when the button created by createButton is clicked.</p> <p>Estimates the size of current content. Use this method with routines that quickly need to know the content size.</p> <p>The true size is the size of the buffer returned when published content is cleaned and removed.</p> <p>Example Below are examples of how the method can be used in ewep.js.</p>	<p>elementName - The name of the HTML element that stores the content. The element name may contain the form name to differentiate elements of the same name in different forms, for example: "frmMain.Content".</p> <p>ContentType - The part of the content to examine. The value can one of these case-insensitive values.</p> <p>"whole" - The whole HTML document.</p> <p>"body" - The body of the content.</p> <p>"text" - The size of the text in the content.</p> <p>Return Value The returned long value is an estimate of the number of characters in the selected content.</p>

Method	Description	Arguments
Examples of how the EstimateContentSize method can be used in ewep.js.		
	<pre> function eWebEditProEditor_save(objValueDestination) { . . . if(!this.isSizeExceeded(this.editor.EstimateContentSize("WHOLE"))) { this.status = EWEP_STATUS_SAVING; var sContent = eval('this.editor.' + this.editorGetMethod + '()'); if (!this.isSizeExceeded(sContent.length)) { objValueDestination.value = sContent; this.status = EWEP_STATUS_SAVED; this.initEvent("onsave"); if (this.raiseEvent("onsave") == false) { return false; } } else { ShowSizeIsTooLarge(this, "save"); return false; } } else { ShowSizeIsTooLarge(this, "save"); return false; } . . } function ShowSizeIsTooLarge(objedit, seventsource) { objedit.status = EWEP_STATUS_SIZEEXCEEDED; objedit.initEvent("onerror"); objedit.event.source = seventsource; if (objedit.raiseEvent("onerror") != false) { if (eWebEditProMessages.sizeExceeded) { alert(eWebEditProMessages.sizeExceeded); } } } </pre>	

Method	Description	Arguments
load()	Loads content into all the in-line editors on the page from the standard HTML elements (typically an <code>input type=hidden</code> field) with the same name.	
refreshStatus()	Updates the values of the following properties: <ul style="list-style-type: none"> ● status ● isIE ● isNetscape ● browserVersion ● isSupported ● isAutoInstallSupported ● isInstalled ● versionInstalled ● upgradeNeeded 	
save()	Saves content from all the in-line editors to the standard HTML elements (typically an <code>input type=hidden</code> field) with the same name.	

Events

The `eWebEditPro.event` object is available during an event. The object's properties are determined by the event.

Event	Occurs when	The <code>eWebEditPro.event</code> object properties
oncreate	The create method is invoked. If the event function returns false, the operation is aborted.	<p>The arguments passed to the <code>create</code> method. You can change the values of these properties in the <code>oncreate</code> event to alter the values used to create an instance of the editor.</p> <p>Refer to the <code>create</code> method for a description of these arguments.</p> <ul style="list-style-type: none"> ● name ● width ● height ● parameters

Event	Occurs when	The eWebEditPro.event object properties
oncreatebutton	The createButton method is invoked. If the event function returns false, the operation is aborted.	<p>The arguments passed to the createButton method.</p> <p>You can change the values of these properties in the oncreatebutton event to alter the values used to create an instance of a popup button to the editor.</p> <p>Refer to the createButton method for a description of these arguments.</p> <ul style="list-style-type: none"> • buttonName • elementName • parameters
onbeforeedit	<p>The onbeforeedit method is invoked. If the event function returns false, the operation is aborted.</p> <p>The onbeforeedit method is called when the user clicks the button created by the createButton method.</p>	<p>The argument passed to the edit method.</p> <p>You can change the value of this property in the onbeforeedit event to change which instance of the editor is opened in the popup window.</p> <p>Refer to the edit method for a description of this argument.</p>
onedit	After the popup window closes.	<p>Indicate which popup editor just closed.</p> <ul style="list-style-type: none"> • elementName - The name of the element that was just edited • popup - A reference to the popup object
onbeforeload	The load method is invoked. If the event function returns false, the operation is aborted.	Undefined
onbeforesave	The save method is invoked. If the event function returns false, the operation is aborted.	Undefined

Event	Occurs when	The eWebEditPro.event object properties
onsave	<p>The save method is complete. If the event function returns a boolean value (true or false), the save method returns the value.</p> <p>A false value can be used to cancel leaving the page in some browsers.</p> <p>The save method is called when the page is unloaded, that is, in the document's onbeforeunload (IE only) or the onunload event, and also on the onsubmit event.</p> <p>Note that the onsubmit event is not fired when the form's submit method is called. It only occurs when the user clicks the submit button.</p> <p>If you are manually calling the submit method, also call the eWebEditPro.save method. The save method is not called on the onunload event if the window.eWebEditProUnloadHandled property is set true prior to calling the create method.</p> <p>You can also prevent VisualFormat from copying content to the hidden field when the onsubmit event fires. To do this, set the <code>document.yourFormsName.eWebEditProSubmitHandled</code> property to true prior to calling the create method.</p>	Undefined.
onload	<p>The load method is complete.</p> <p>The load method is called when the page is loaded, that is, in the document's onload event.</p> <p>The load method is not called if the window.eWebEditProLoadHandled property is set to true prior to calling the create method.</p>	Undefined

Event	Occurs when	The eWebEditPro.event object properties
onready	<p>It is safe to send commands or access the MediaFile object.</p> <p>For example</p> <pre data-bbox="392 342 809 612"> eWebEditPro.onready = "initTransferMethod(eWebEditPro.event .srcName)"; function initTransferMethod(strEditorName) { eWebEditPro[strEditorName].MediaFile().setProperty("TransferMethod", "mediamanager.asp"); } </pre>	<ul style="list-style-type: none"> • type - Ready • srcName - The name of the instance of the editor that is the source of the current event.
onerror	<p>An error occurs because the save method failed. Inspect the status property to determine the cause of the error.</p> <p>Error codes include:</p> <ul style="list-style-type: none"> • EWEP_STATUS_NOTINSTALLED • EWEP_STATUS_NOTLOADED • EWEP_STATUS_UNABLETOSAVE • EWEP_STATUS_SIZEEXCEEDED • EWEP_STATUS_FATALError 	<p>Provide information about the source and cause of the error.</p> <ul style="list-style-type: none"> • source - The name of the method that is the source of the error. For example, "load" if the load method fails. • name - The name of the editor, where name is the argument passed to the create method. • instance - A reference to the instance object.

The Event Object

When an ActiveX control fires, the **event** object in the VisualFormat JavaScript object contains properties that describe the source of the event.

Property	Description	Example
type	The name of the current event without the "on" prefix. The type is always lowercase.	<pre data-bbox="752 1315 1305 1406"> if (eWebEditPro.event.type == "dblclickelement") { ... } </pre>
srcName	The name of the instance of the editor that is the source of the current event.	<pre data-bbox="752 1439 1324 1645"> onDoubleClickElementHandler(oElement) { // Replace element that was double-clicked with a // horz line. eWebEditPro.instances[eWebEditPro.event.srcName].ed itor.pasteHTML("<hr>"); or eWebeditPro[eWebeditPro.event.srcName].pasteHTML("< hr>"); } </pre>

Event Handler Functions

Event Handler Function	Description	Arguments
eWebEditProExecCommand	<p>The JavaScript defined in this function is called after an internal command is executed, or when an external command should be executed. That is, when a toolbar button is pressed or a command is selected, such as on the context menu or drop-down list on the toolbar.</p> <p>Writing the function eWebEditProExecCommand is the preferred way to add custom commands rather than modifying onExecCommandDeferred or onExecCommandHandler in ewebeditproevents.js.</p> <p>Return true to allow the default external commands to run (see eWebEditProExecCommandDispatcher).</p> <p>Return false to prevent default external commands from running. Internally handled commands will have already been executed prior to this event firing.</p>	<p>(sEditorName, strCmdName, strTextData, IData)</p> <ul style="list-style-type: none">● sEditorName - the name of the version of VisualFormat whose command was executed (for example, "MyContent1"). To access the VisualFormat methods, use eWebEditPro[sEditorName]. See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.● strCmdName - a string containing the command, for example, "cmdCut"● strTextData - a string that may contain text data related to the command. Typically not used.● IData - a long integer value that may contain numeric data related to the command. Typically not used. <p>The IData parameter does not reflect the index of the list box item. Instead, it only returns the data assigned to the item.</p> <p>If, in the processing of the command notification, you need to retrieve the index of the selected item, use objCommand.getPropertyInteger("CmdIndex").</p>

Event Handler Function	Description	Arguments
<p>eWebEditProReady or, eWebEditPro.onready event</p> <p>where the event object has these properties:</p> <ul style="list-style-type: none"> ● type = "ready" ● srcName = name of the editor that is ready 	<p>Indicates when it is safe to send commands or access the MediaFile object.</p>	<p>(strEditorName) - a string containing the name of the editor of interest, for example, "MyContent1."</p> <p>See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.</p> <p>Example</p> <pre>eWebEditPro.onready = "initTransferMethod(eWebEditPro.event.srcName);" function initTransferMethod(strEditorName) { eWebEditPro[strEditorName].MediaFile().setProperty("TransferMethod", "mediamanager.asp"); }</pre>
eWebEditProMediaSelection	<p>To add your own media file handler, define this JavaScript function in your web page. This function is called when the picture button is pressed.</p>	<p>(sEditorName)</p> <p>sEditorName is the name of the VisualFormat editor whose command was executed.</p> <p>To access VisualFormat methods, use</p> <pre>eWebEditPro[sEditorName].</pre> <p>See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.</p>
Double-Click Element Handlers		
<p>To add your own double-click element handler, define a JavaScript function in your web page to run as shown below. The eWebEditProDblClickElement function runs when certain elements are double-clicked. It may be easier, however, to define the applicable handler function for a specific object.</p> <p>The hyperlink, image, and table element objects have their own functions that run when they are double-clicked.</p>		
<pre>eWebEditProDblClickElement(oElement) { return true or false }</pre>	<p>The name of the JavaScript event that occurs when a user double-clicks a hyperlink, applet, object, image or table within the editor, unless a specific event handler for hyperlink, image or table is defined.</p> <p>The default event handlers are defined in the ewebeditproevents.js file.</p>	<p>(oElement)</p> <p><i>oElement</i> is a reference to the HTML element that was double-clicked.</p> <p>Return</p> <ul style="list-style-type: none"> ● true to allow the default external commands to run (see eWebEditProExecCommandDispatcher) ● false to prevent them from running <p>Note that internally handled commands will have been executed prior to this event firing.</p>

Event Handler Function	Description	Arguments
<pre>eWebEditProDblClickHyperlink(oElement) { return true or false }</pre>	<p>The name of the JavaScript event that occurs when a user double-clicks a hyperlink.</p> <p>The default hyperlink event handler is defined in the ewebeditproevents.js file.</p>	see argument information for eWebEditProDblClickElement
<pre>eWebEditProDblClickImage(oEl ement) { return true or false }</pre>	<p>The name of the JavaScript event that occurs when a user double-clicks an image.</p>	see argument information for eWebEditProDblClickElement
<pre>eWebEditProDblClickTable(oEle ment) { return true or false }</pre>	<p>The name of the JavaScript event that occurs when a user double-clicks a table.</p>	see argument information for eWebEditProDblClickElement

Reacting to the Creation of a Toolbar

You can use the toolbarreset command to reset toolbar features using a script. The script can either implement the toolbarreset command or react to the command.

Script Implementing a Toolbarreset Command

The toolbarreset command can be sent by a script or defined in the configuration data when you want to quickly reset the toolbar features.

Script Reacting to a Toolbarreset Command

The toolbarreset command can be sent to a script as a command passed in the eWebEditProExecCommand function. If the scripting adds any buttons to the toolbar, the new toolbar configuration is saved in the user's customization settings, if allowed. (See "Allowing Users to Customize the VisualFormat Toolbar" on page 137.)

Here is an example of how a script can react to the command.

```
function eWebEditProExecCommand(sEditorName, strCmdName, strTextData, lData)
{
    if("toolbarreset" == strCmdName)
    {
        var objInstance = eWebEditPro.instances[sEditorName];
        var objMenu = objInstance.editor.Toolbars();

        // This shows how to create a menu.
        objMenu.ToolbarAdd("SampleMenu", "This is a Script Menu ", 0, 0, 0, 0, "");

        // This shows how to add commands to a menu.
        objMenu.CommandAdd("jssample1", "Sample Command 1", "Sample Command 1",
            "key", 0, 0, "SampleMenu", 0, -1);
        objMenu.CommandAdd("jssample2", "Sample Command 2", "Sample Command 2",
            "key", 0, 0, "SampleMenu", 0, -1);
        objMenu.CommandAdd("jssample3", "Sample Command 3", "Sample Command 3",
            "key", 0, 0, "SampleMenu", 0, -1);
        objMenu.CommandAdd("jssample4", "Sample Command 4", "Sample Command 4",
            "key", 0, 0, "SampleMenu", 0, -1);
        objMenu.CommandAdd("jssample5", "Sample Command 5", "Sample Command 5",
            "key", 0, 0, "SampleMenu", 0, -1);

        if("jssample1" == strCmdName)
        {
            alert("Hello from sample command 1");
        }
        if("jssample2" == strCmdName)
        {
            alert("Hello from sample command 2");
        }
        if("jssample3" == strCmdName)
        {
            alert("Hello from sample command 3");
        }
        if("jssample4" == strCmdName)
        {
            alert("Hello from sample command 4");
        }
        if("jssample5" == strCmdName)
        {
            alert("Hello from sample command 5");
        }
    }
}
```

When the Command is Sent to the Script

The toolbarreset command is sent under either of these conditions.

- A new toolbar is loaded. If no saved customization is found, the configuration data is read to build a fresh toolbar. Next, the toolbarreset command is sent to allow the script to add commands to the toolbar.
- The toolbars are reset. When this happens, all old customizations are discarded, and the configuration data is read to create a new toolbar. At this time, the script can add commands by reacting to the toolbarreset command.

Command Parameters

The toolbarreset command has two parameters.

Parameter	Type	Value if toolbar is freshly loaded from config.xml, not loaded from a saved configuration	Value if toolbar is generated by a reset
strTextData	string	NewLoad	FullReset
lData	long	1	0

Implementing toolbarreset as a Toolbar Button

In the configuration data, you can assign a button to the toolbarreset command. If you do, the command executes when the user clicks the button. The command is passed on to the script after it executes, just as with standard commands.

No icon is assigned to the command, so you can choose any standard image. (See "Button Images" on page 179.)

WARNING!

The toolbarreset would be a dangerous toolbar button. The user could accidentally click it and reset everything.

You may prefer to define your own external command, such as "jstoolbarreset", and implement it in the scripting instead of defining the command as a standard button. In this way, you can interact with the user appropriately and then send the "toolbarreset" command to the editor.

Using Toolbarreset to Reset Customization

You can use the toolbarreset command to reset a user's customization to a new XML configuration definition, even without changing the customization name in the configuration data.

To do this, save a cookie to the user's system. The script can check to see if the customization has happened. If it has not, the script could call the reset command to reset the named customization to the new XML definition. Then, the script could use the cookie to record that the update was done.

The Redisplay Toolbars Command

Name: redisplaytoolbars

Parameters: None

Description: Displays, or unhides, all toolbars. This command is only useful if the user removes all menus and cannot customize to get them back.

This command appears as the **Restore All Menus** item on the context menu *only* if the user cannot customize toolbars and the user has removed all menus from view.

See Also: "Allowing Users to Customize the VisualFormat Toolbar" on page 137.

The Instance Object

The instance object is accessed via the instances collection of the eWebEditPro object. For example:

```
var objInstance = eWebEditPro.instances[ "MyContent1" ]
```

Properties

Property	Effect of Response
editor	A reference to the VisualFormat ActiveX control. For example eWebEditPro.Editor1.pasteHTML("<HR>") is equivalent to eWebEditPro.instances["Editor1"].editor.pasteHTML("<HR>") Read-only.
elemName	The name of the field element that contains the editor content. This is typically the name specified when creating the editor. See Also: "formName" on page 103
name	The name assigned to this instance of the editor when it was created. Read-only.
receivedEvent	This boolean value is true if an event has been received from the ActiveX control. This property is used internally and is for reference only. It is not necessary for typical development.

Property	Effect of Response
status	The status of this editor. (It has the same status value as the VisualFormat JavaScript object, but only applies to this instance of the editor). Do not change. The status of the VisualFormat JavaScript object is described in "status" on page 88.
width	The width of the editor assigned when created. Read-only.
formName	The name or index of the form that contains this instance of the editor. See Also: "elemName" on page 102 Example: <pre>function myOnEventHandler() { var objInstance = eWebEditPro.instances[eWebEditPro.event.srcName]; var strContent = document.forms[objInstance.formName].elements[objInstance.elemName].value; }</pre>
height	The height of the editor assigned when created. Read-only.
html	A string containing the HTML. To create the editor in a window other than the current one, set eWebEditPro.parameters.editorWindow to the name of the window. For example <pre><script> frame2, document, open(0); eWebEditPro.parameters.editorWindow="frame2"; eWebEditPro.create(...); frame2.document.close(); </script></pre> To prevent the editor from writing the HTML to the window document, set eWebEditPro.parameters.writeDisabled to true . For example, <pre><script> eWebEditPro.parameters.writeDisabled="true"; var Obj Editor=eWebEditPro.create(...); var strHTML=Obj.html; </script></pre>
id	The name of the VisualFormat editor element in the object (Internet Explorer) or embed (Netscape) tag. Typically not used. Read-only.
maxContentSize	See "maxContentSize" on page 108.

Methods

Method	Description
insertMediaFile (strSrcFileLocation, bLocalFile, strFileType, nWidth, nHeight)	<p>Inserts an image file (or other media file) to the editor. For images, the IMG tag is used.</p> <p>This method sets properties in the ActiveX control's MediaFile object (see "The Mediafiles Feature of the XML Configuration Data" on page 256) and then executes the cmdmfuiinsert command.</p> <p>Arguments to the insertMediaFile method:</p> <ul style="list-style-type: none">● strSrcFileLocation - the path to file being inserted. The path can be the full path or relative to the host name. If a relative path, the editor uses the current page location/BaseURL to determine the file's location.● bLocalFile - "true" if the file is on the user's computer; "false" if the file is on the server.● strFileType - the image title; if one is not passed, the user must enter one in the Title field of the Image Selection Screen. It is used as the image's alt text.● strFileType - The type of file: image or other.● nWidth - the width of the image in pixels (if the file is an image)● nHeight - the height of the image in pixels (if the file is an image) <p>Example</p> <pre>eWebEditPro.instances["MyContent1"].insertMediaFile("mypic.jpg", false, "My Picture Title", image, 80, 60);</pre>
load(valueSource)	<p>Loads content into editor. Not typically needed.</p> <p>valueSource may be</p> <ul style="list-style-type: none">● undefined (content is loaded from the content element)● an object with a 'value' property● a string
save(objValueDestination)	<p>Saves content. Not typically needed.</p> <p>objValueDestination may be</p> <ul style="list-style-type: none">● undefined (content is stored to the content element)● an object (the value property will be set)

Events

The following events function as they do with the VisualFormat object, but only apply to this instance of the editor. VisualFormat object events are described in “Events” on page 93.

- onbeforeload
- onload
- onbeforesave
- onsave
- onerror

The onerror Event

Note that, for the instance object’s onerror event,

- usually only the source property is available
- the event only fires if the save method fails

If the status of the VisualFormat object is EWEP_STATUS_SIZEEXCEEDED, two additional event properties are available to help troubleshoot the error.

- **contentSize** - the number of characters in the content
- **maxContentSize** - the maximum number of characters permitted, as specified by the maxContentSize parameter (See “maxContentSize” on page 108.)

Here is an example of how to use these properties.

```
function myErrorHandler()
{
    if (EWEP_STATUS_SIZEEXCEEDED == this.status && "save" ==
this.event.source)
    {
        var strMsg = "HTML content size (in chars): " +
this.event.contentSize + " Maximum: " + this.event.maxContentSize;
        alert(strMsg);
    }
}

eWebEditPro.instances.MyContent1.onerror = myErrorHandler; //
```

Note that you cannot use ...onerror =
"myErrorHandler()"; .

The Parameters Object

The parameters object is used to set parameters prior to creating an instance of the editor. This is a property of the eWebEditPro object (for example, eWebEditPro.parameters).

Use the parameters object to change default values for a particular instance of an editor. To change the default values for all instances of the editor, change the value in ewebeditprodefaults.js.

The names of most parameters match the names in defaults.js. The different ones are listed below.

NOTE It is important to retain the case (upper or lower) of the letters when changing a parameter value.

default.js	parameters.
buttonTagStart	.buttonTag.start
buttonTagEnd	.buttonTag.end
popup*	.popup.* (the first letter after a popup becomes lowercase)

For a complete list of parameters and their corresponding default.js values, see “Customizing the Popup Button” on page 43.

Properties

These properties are the same as those in ewebeditprodefaults.js. In fact, ewebeditprodefaults initializes the parameters object.

The following parameters are part of the ActiveX control. Go to the listed page numbers to read about them.

- "hideAboutButton" on page 118
- "Config" on page 118
- "BaseUrl" on page 116
- "CharSet" on page 118

-
- "Title" on page 119
 - "bodyStyle" on page 117

Since the following properties are not part of the ActiveX control, they are documented below.

NOTE

Onblur, ondblclickelement, onexeccommand, and onfocus are events raised by the ActiveX control, not the parameters object. But you set them using the parameters object -- you cannot set them using the ActiveX control. As a result, they are documented as properties of the parameters object used to assign JavaScript that executes when the ActiveX control's event fires..

Property	Description
buttonTag	Object consisting of <ul style="list-style-type: none"> • eWebEditProDefaults.buttonTagStart • eWebEditProDefaults.buttonTagEnd • eWebEditProMessages.popupButtonCaption If null, the popup button does not appear.
clientInstall	The directory in which the client installation file (ewebeditproclient.exe) resides.
cols	The number of columns in the TEXTAREA element if VisualFormat is not installed or not supported. If undefined, the number of columns approximates the width specified when the browser is created.
editorGetMethod	Lets you save either the body only or the entire HTML document from the editor. You can set this method in ewebeditprodefaults.js by editing this line: <pre>this.editorGetMethod = "getBodyHTML"; // "getBodyHTML" or "getDocument"</pre> Or, you can set this method directly in the web page that calls the editor using the following JavaScript: <pre>eWebEditPro.parameters.editorGetMethod = "value"</pre> The possible values are getBodyHTML (see "getBodyHTML()" on page 121) and getDocument (see "getDocument()" on page 121).
embedAttributes	Optional attributes to the EMBED tag. Applies only to Netscape.

Property	Description
maxContentSize	<p>The largest number of characters that can be saved in the editor window. If a user enters content that exceeds this size, an error message appears.</p> <p>The maximum size of the content may be increased in some circumstances. Several factors affect the maximum size allowed.</p> <ul style="list-style-type: none"> ● Netscape 4 fields are limited to 64K, that is 65535. ● The web server may limit the size of form variables (e.g., hidden fields), although typically the size is very large. ● If using a database, the database field type may be limited in size (e.g., 64K bytes). Check your database documentation. ● If using ODBC, the ODBC driver on the server may limit the content. <p>Also, you may want to limit content size as a matter of corporate policy, personal preference, or to implement quotas where a user has a limited amount of space allocated.</p> <p>If none of these restrictions applies to your situation (for example, all users have Internet Explorer), you can increase the value of maxContentSize in ewebeditprodefaults.js or set it in JavaScript.</p> <p>To have no limit, set maxContentSize = 0.</p> <p>Note: This parameter checks the number of characters, which may be different from the number of bytes, depending on the encoding method.</p>
objectAttributes	<p>Optional attributes to the OBJECT tag. Applies only to Internet Explorer.</p> <p>For example, the OBJECT tag has an attribute 'standby' that the developer could set to a string.</p> <pre>objectAttributes="standby='Please wait while the editor initializes.'";</pre>
onblur	<p>An event that fires when the editor loses the focus. onblur is a standard DHTML event.</p>
ondblclickelement	<p>The JavaScript event that occurs when a user double-clicks any selectable element object. Objects include hyperlink, applet, image, table, form element, and any absolutely positioned element (for example, <code><div style="position: absolute"></div></code>) within the editor.</p> <p>The default event handlers are defined in the ewebeditproevents.js file.</p> <p>Arguments passed to the ondblclickelement handler function must be all lowercase.</p> <p>See also "ActiveX Events" on page 131.</p>

Property	Description
onexeccommand	<p>The default JavaScript onexeccommand handler.</p> <p>The onexeccommand event fires when a command is executed, such as when a toolbar button is pressed or a context menu item is selected.</p> <p>The onexeccommand event is used to detect when standard VisualFormat commands are executed (the event fires <i>after</i> the command), and is also used to implement custom commands that you define.</p> <p>Since you cannot access the ActiveX control directly from within an event handler, the default handler (defined in ewebeditproevents.js) calls setTimeout() to defer the execution of the event handler.</p>
onfocus	<p>An event that fires when the editor gains the focus. onfocus is a standard DHTML event.</p>
path	<p>The path to the VisualFormatfiles relative to the hostname. For example, /ewebeditpro2/.</p> <p>This value is set in the ewebeditpro.js file.</p>
Installation Popup Window Defaults	<p>This set of defaults determines the attributes of the instance of VisualFormat that appears as a popup window.</p>
installPopup.url	<p>Specifies the URL of a web page to display in a popup window when an automatic installation is expected.</p> <p>Example in ewebeditprodefault.js</p> <pre>this.installPopupUrl = this.path + "clientinstall/intro.htm";</pre> <p>See also:</p> <ul style="list-style-type: none"> ● “Client Installation Pages” on page 84. ● “installPopup.open()” on page 114
installPopup.windowName	<p>Specifies the name of the popup window. Typically, this is left as an empty string.</p> <p>Example in ewebeditprodefault.js:</p> <pre>this.installPopupWindowName = "";</pre>
installPopup.windowFeatures	<p>Specifies the popup window features as defined for the standard JavaScript window.open() method.</p> <p>Example in ewebeditprodefault.js</p> <pre>this.installPopupWindowFeatures = "height=540,width=680,resizable,scrollbars,status";</pre> <p>See Also: “popup.windowFeatures” on page 113.</p> <p>For more details on the JavaScript window.open() method, see a JavaScript reference.</p>

Property	Description
installPopup.query	<p>An optional parameter that specifies query string values to pass to the page specified by the URL parameter. Typically, the query property is left as an empty string.</p> <p>If specified, the query string is appended to the URL, separated by a question mark (?) character. Do not include the ? in the query string value.</p> <p>Example in ewebeditprodefault.js</p> <pre>this.installPopupQuery = "";</pre> <p>Example in JavaScript</p> <pre>eWebEditPro.parameters.installPopup.query = "firstname=Bob&lastname=Smith";</pre>
Popup Window Defaults: buttonTag.start, buttonTag.end, buttonTag.value	<p>This set of defaults determines the attributes of the button that launches the popup window.</p> <p>Examples</p> <pre><script language="JavaScript"> var strhref = "javascript:"; if (eWebEditPro.isNetscape) { strhref += "eWebEditPro.edit(\"MyContent1\")"; } eWebEditPro.parameters.buttonTag.start = "Edit width=150 height=60 src=button.gif"; eWebEditPro.parameters.buttonTag.value = ""; eWebEditPro.parameters.buttonTag.end = ">"; </script></pre>

Property	Description
popup	<p>Lets you pass four parameters to the popup web page (specified in <code>popupUrl</code> property).</p> <ul style="list-style-type: none"> ● <code>url</code> (see “<code>popup.url</code>” on page 112) ● <code>windowName</code> (see “<code>popup.windowName</code>” on page 113) ● <code>windowFeatures</code> (see “<code>popup.windowFeatures</code>” on page 113) ● <code>query</code> (see “<code>popup.query</code>” on page 112) <p>By default, the <code>popupUrl</code> page is a static HTML page, but it could be a dynamically generated page. In either case, you may want to pass additional information to the popup page. For example, you may want to display the number of times the content has been edited, the title of the content, or anything else.</p> <p>Here is an example that passes a title and instructions relevant to the content being edited.</p> <p>On the page with the popup button:</p> <pre> <script language="JavaScript"> var sTitle = "Summary Description"; var sInstr = "Please enter a paragraph summarizing the page."; with (eWebEditPro.parameters.popup) { url = "cif_t0007popup.htm"; windowName = ""; windowFeatures = "location,scrollbars,resizable"; query ="title=" + escape(sTitle) + "&instr=" + escape(sInstr); ... eWebEditPro.createButton(. . .); </script></pre> <p>(The JavaScript <code>escape()</code> function ensures the text is saved to pass in a URL. For example, it changes all space characters to %20. The <code>unescape()</code> function restores the text.)</p>

continued on next page

Property	Description
popup (continued from previous page)	<p>On the popup page:</p> <pre data-bbox="507 238 1338 895"><script language="JavaScript"> var objQuery = new Object(); var strQuery = location.search.substring(1); var aryQuery = strQuery.split("&"); var pair = []; for (var i = 0; i < aryQuery.length; i++) { pair = aryQuery[i].split("="); if (pair.length == 2) { objQuery[unescape(pair[0])] = unescape(pair[1]); } } document.writeln("<p>" + objQuery["title"] + "</p>"); document.writeln(objQuery["instr"] + "
"); </script></pre>
popup.url	<p>The URL to the web page that contains the editor that appears in the popup window. The default value is</p> <pre data-bbox="507 968 1065 995">this.path + "ewebeditpropopup.htm";</pre>
popup.query	<p>Enter a query to pass parameters to the popup window.</p> <hr/> <p>Note: The popup.query property must not include the question mark (?) character.</p> <hr/>

Property	Description
popup.windowFeatures	<p>The parameters passed to the standard JavaScript <code>window.open()</code> method.</p> <p>To enable a feature (e.g., scrollbars), include the keyword. To disable a feature, exclude the keyword.</p> <p>Separate each feature keyword by a comma, but <i>include no spaces</i> between parameters.</p> <p>A few of the possible features include:</p> <ul style="list-style-type: none"> ● <code>width=x</code>, where <code>x</code> is the window width in pixels ● <code>height=y</code>, where <code>y</code> is the window height in pixels ● <code>scrollbars</code>: displays scrollbars ● <code>status</code>: displays the status bar ● <code>resizable</code>: the user can change the window size ● <code>location</code>: displays the location (or address) bar ● <code>menubar</code>: displays the menu bar ● <code>toolbar</code>: displays toolbar buttons <p>For more details on the JavaScript <code>window.open()</code> method, see a JavaScript reference.</p>
popup.windowName	<p>The name assigned to the popup window created by the standard JavaScript function <code>window.open()</code>.</p>
rows	<p>The number of rows in the TEXTAREA element if VisualFormat is not installed or not supported.</p> <p>If undefined, the number of rows approximates the height specified when the editor is created.</p>
textareaAttributes	<p>Optional attributes to the TEXTAREA tag. Apply only when a textarea field appears in place of VisualFormat, typically because the operating system does not support VisualFormat.</p> <p>You can specify the row and column attributes of the textarea field using the rows and cols parameters. For example, you could use the <code>textareaAttributes</code> to specify an <code>onchange</code> attribute value.</p> <p>For example,</p> <pre>textareaAttributes = "onchange='mychangehandler()'" ;</pre>

Methods

Method	Description
installPopup.open()	Displays the page specified by the installPopup parameters in a popup window.
installPopup.close()	Closes the popup window.
reset()	<p>Reinitializes all values to the default defined in eWebEditProDefaults (ewebeditprodefaults.js). This method should be called after creating an editor if properties were changed for that instance of the editor.</p> <p>If reset() is not called, any changed property values apply to all subsequent instances of the editor.</p>

ActiveX Control

Web masters can exert control over VisualFormat's functionality and content through modifying the ActiveX control properties and methods.

This chapter explains the properties, methods and events of the VisualFormat ActiveX control.

Accessing the ActiveX Control

To access the ActiveX control via JavaScript, once an instance of the editor is created, use the following VisualFormat JavaScript object.

```
eWebEditPro.theeditorname
```

The ActiveX control should not be accessed until after the VisualFormat onload event fires (see "onload" on page 95).

For example

```
eWebEditPro.Editor1.pasteHTML( "<HR>" ); // insert horz rule
```

The pasteHTML method inserts HTML content into the editor. For more information, see "pasteHTML(sHTMLText)" on page 122.

ActiveX Properties

You can modify the values for the default ActiveX control properties in the ewebeditprodefaults.js file, using a standard text editor such as Notepad.

You can also modify ActiveX control property values for individual instances of the editor. See "The Parameters Object" on page 106.

The following table explains VisualFormat's ActiveX control properties.

ActiveX Control Property	Specifies
BaseUrl	<p>Type: String</p> <p>This property sets the current URL of the editor to the specified location. All references are based on, and relative to, this location. The control uses this value to find items to display.</p> <p>This property does <i>not</i> need to be set. If it is not set, the ActiveX control determines its current location. The property is useful for allowing a script to point the editor at another location.</p> <p>IMPORTANT: A trailing slash is required.</p> <p>Also, if image paths are relative, you must set the <code>xferdir</code> and <code>webroot</code> attributes of the <code>mediafiles</code> element to match this value. Otherwise, the transfer and reference directories may not be on the same server, and the current URL defaults to a full path resolution.</p> <p>Example:</p> <p>Change to match the setting of the BaseURL in the <code>visualformatconfig.xml</code> data.</p> <pre data-bbox="389 707 950 1096"><features> ... <mediafiles> ... <transport> ... <xferdir src="http://msimg.com/w"/> <!-- set to new location --> <webroot src="" /> <!-- Keep blank so it matches xferdir --> </transport> </mediafiles> </features></pre> <p>Example addition in the script that can change this value.</p> <pre data-bbox="389 1090 1291 1187">function myOnReady(sEditorName) { eWebEditPro.instances[sEditorName].setProperty("BaseUrl", "http://msimg.com/w"); }</pre>

ActiveX Control Property	Specifies
bodyStyle	<p>Set cascading style sheet (CSS) attribute values, such as background color, default font style, size, color and more. The bodyStyle parameter sets any valid CSS style supported by your browser.</p> <p>Note that this property affects the same values as the style attribute in the body tag, for example:</p> <pre><body style="background-color: black; font-size: 24pt"></pre> <p>If both are set, the bodyStyle parameter takes precedence.</p> <p>Effect of Style Sheet on bodyStyle Parameter</p> <p>If a style sheet is being referenced by the editor, the style sheet's specifications override any font styles defined in the bodyStyle parameter except for the BODY element. So, for example, while you can set the default font using bodyStyle, that setting will have no effect on text within the <P> tag.</p> <p>If a style sheet is being used, you have three options for resolving the clash of style specifications between the bodyStyle parameter and the style sheet.</p> <ul style="list-style-type: none"> • disable the style sheet • change the style sheet so that it specifies the desired default font • change the style sheet so that it does not specify font attributes for the content. The style sheet can continue to specify other attributes, such as page break after, widow/orphan control, and margins <p>When to Set the Parameter</p> <p>This parameter may be set before or after the editor is created. It may also be changed while the editor is loading, and the change will apply immediately.</p> <p>Examples</p> <p>Examples of how to set the body style property appear below.</p> <p>In ewebeditprodefaults.js -</p> <pre>this.bodyStyle = "background-color: black; font-size: 24pt";</pre> <p>JavaScript parameter before the editor is created -</p> <pre>eWebEditPro.parameters.bodyStyle = "background-color: black; font-size: 24pt";</pre> <p style="text-align: right;"><i>continued on next page</i></p>

ActiveX Control Property	Specifies																
bodyStyle (continued)	<p>ActiveX control property after the editor is created -</p> <pre data-bbox="392 269 1262 311"><input type=button value="Red Background" onclick="eWebEditPro[MyContent1].setProperty("BodyStyle" , "background:red")"></pre> <p>Specifying Font Size</p> <p>In version 1.8, the fontSize parameter specified the value for the font tag's size attribute. The valid values were 1 through 7.</p> <p>For CSS, however, it is customary to specify size in points.</p> <p>The table below maps font sizes to point sizes.</p> <table data-bbox="392 497 605 786"> <thead> <tr> <th data-bbox="392 497 432 521">fontSize</th> <th data-bbox="432 497 605 521">Point Size</th> </tr> </thead> <tbody> <tr> <td data-bbox="392 536 418 559">1</td> <td data-bbox="418 536 533 559">.....8pt</td> </tr> <tr> <td data-bbox="392 572 418 596">2</td> <td data-bbox="418 572 533 596">.....10pt</td> </tr> <tr> <td data-bbox="392 609 418 632">3</td> <td data-bbox="418 609 533 632">.....12pt</td> </tr> <tr> <td data-bbox="392 645 418 669">4</td> <td data-bbox="418 645 533 669">.....14pt</td> </tr> <tr> <td data-bbox="392 682 418 705">5</td> <td data-bbox="418 682 533 705">.....18pt</td> </tr> <tr> <td data-bbox="392 718 418 742">6</td> <td data-bbox="418 718 533 742">.....24pt</td> </tr> <tr> <td data-bbox="392 755 418 778">7</td> <td data-bbox="418 755 533 778">.....36pt</td> </tr> </tbody> </table> <p>IMPORTANT: Setting the bodyStyle only applies to content in the editor. When the content of the body is saved, the body style attribute is not saved (unless you save the whole document).</p> <p>You need to also apply the style to the web page that displays the content, such as a template file in a template-based content management system.</p>	fontSize	Point Size	18pt	210pt	312pt	414pt	518pt	624pt	736pt
fontSize	Point Size																
18pt																
210pt																
312pt																
414pt																
518pt																
624pt																
736pt																
CharSet	<p>The charset value for a page. For example</p> <pre data-bbox="392 1048 921 1072"><meta content="text/html; charset=iso-8859-1">.</pre> <p>This is only needed if saving the entire document.</p>																
Config	<p>The URL of the config XML file. Although this ActiveX control property can contain the XML content, it typically refers to an XML file. (For details, see "Managing the Configuration Data" on page 134.)</p>																
hideAboutButton	<p>Set to "True" to remove the About () button from the toolbar.</p>																
License	<p>The license keys of the editor. Separate each with a comma.</p> <p>Ektron provides these keys after purchase. For development purposes, the license keys for 127.0.0.1 and localhost are built into the editor.</p> <p>Note: VisualFormat displays an "Invalid License" message if the license key is improperly entered.</p>																
Locale	<p>The URL of the localization directory or file. (For details, see "Modifying the Language of VisualFormat" on page 70.)</p>																

ActiveX Control Property	Specifies
srcPath	<p>Where VisualFormat is installed, that is, the eWebEditProPath. The XML configuration data has an environment variable [eWebEditProPath], which is replaced by the value in srcPath.</p> <p>Do not change the value of srcPath.</p>
StyleSheet	<p>Which style sheet file (CSS) to apply to the editor content. If the entire document is retrieved from the editor, the style sheet is included in the head section using the link element.</p> <p>The value of the StyleSheet property is the value of the link tag's href attribute.</p> <p>Examples</p> <ul style="list-style-type: none"> • via parameter before the editor is created: <pre>eWebEditPro.parameters.styleSheet = "mystyle.css";</pre> <ul style="list-style-type: none"> • via ActiveX after the editor is loaded and ready: <pre>eWebEditPro.myEditor1.setProperty("StyleSheet", "mystyle.css");</pre> <p>Both examples produce this line between the document's head tags:</p> <pre><link rel="stylesheet" type="text/css" href="mystyle.css"></pre> <p>If only the content within the body tags is retrieved, you need to also apply the style sheet to the HTML document that displays the content.</p> <p>If you change the StyleSheet property, the changes are visible immediately.</p> <p><i>See Also:</i> "Style Sheets" on page 219</p>
Title	<p>A document title for the page. For example</p> <pre><head> <title>Ektron, Inc. - dynamic web content management - html editor</title></pre> <p>This is only needed if saving the entire document.</p>

ActiveX Control Property	Specifies
Version	<p>This property retrieves the version of the control. It is a comma delimited list with this format:</p> <p>Major Major, Minor Major, Major Minor, Minor Minor</p> <p>Or</p> <p>Version, Point Release, just 0, Revision</p> <p>(The Major Minor value is not used, so it is always 0.)</p> <p>Syntax</p> <p>strData = [form!]eWebEditPro2.Version</p> <p>Remarks</p> <p>The Major Minor value of 0 is in the format because of the agreed upon format for software module versions. If comparing versions, the string must be parsed and each item converted to an integer.</p> <p>Examples</p> <p>Displays the control version:</p> <pre>function ShowVersion() { alert(testItem1.Version); } or alert(eWebEditPro.MyContent1.version); or alert(eWebEditPro.MyContent1.getPropertyString("version"));</pre>

ActiveX Methods

Method	Description
ExecCommand(strCmdName, strTextData, IData)	Causes the editor to perform the specified operation. For more information, see "Using JavaScript to Send Commands" on page 24.
getBodyHTML()	Saves the content within the BODY tags as HTML. The HTML will be a valid fragment.
getBodyText()	<p>Returns the content text without formatting. Note that <i>only</i> the text is returned, not the html code.</p> <p>This method is used by browser-based email applications that need both content with HTML tags and content that is text only.</p> <p>To use this method, first add a hidden field to post the text to the server. Then, when the content is saved, copy the text from the editor into the hidden field.</p> <p>These steps illustrate how to use this method.</p> <ol style="list-style-type: none">1. Add a hidden field to store the text.<p>For example,</p><pre><input type=hidden name="MyContentText1" value=""></pre>2. Add JavaScript to copy the text to the hidden field. Use the eWebEditPro.onsave event. This event fires when the content is saved, that is, copied from the editor to the hidden content field.<p>For example, if formName is the name of your form and MyContent1 is the name of the eWebEditPro editor, use this code.</p><pre><script language="JavaScript1.2"> eWebEditPro.onsave = "document.formName.MyContentText1.value = eWebEditPro.MyContent1.getBodyText()"; </script></pre>3. Modify your server-side code to process the text. You may wish to save it in a database field for text searches without the HTML tags. Alternatively, you may wish to email the text to clients with text-only viewers.
getDocument()	Saves the entire HTML document that is currently in the editor.
getHeadHTML()	Returns the <HEAD> through </HEAD> HTML of the current document as a string, inclusive of the HEAD tags. <p>Syntax</p> <pre>strHead = eWebEditPro.Editor1.getHeadHTML</pre> <p>Example</p> <pre>eWebEditPro1.Editor1.getHeadHTML returns <HEAD><TITLE>eWebEditPro Test Page</TITLE> </HEAD></pre>

Method	Description
getProperty (name of property)	Reads from the ActiveX control property.
getPropertyBoolean (name of property)	Returns the value of a Boolean property.
getPropertyInteger (name of property)	Returns the value of a Numeric property.
getPropertyString (name of property)	Returns the value of a String property.
getSelectedHTML()	<p>Returns the currently selected content including any HTML tags. The HTML will be a valid fragment.</p> <p>Pasting the content back into the editor may cause side effects. For example, selecting part of a table returns any HTML tags for a complete table. Pasting it back will insert a table within the table.</p>
getSelectedText()	Returns the currently selected text with no formatting. Only the text is returned, not the html code.
pasteHTML(sHTMLText)	<p>pasteHTML replaces the selected content in VisualFormat with the string passed to pasteHTML.</p> <p>sHTMLText: the content to be pasted into the editor's content at the current cursor location. Any editor content that is selected when pasteHTML is executed is replaced. For example</p> <pre>eWebEditPro.Editor1.pasteHTML("<hr>
Hello World!");</pre>
pasteText(sText)	<p>pasteText replaces the selected content in VisualFormat with the string passed to pasteText. The content is pasted as is: HTML tags are not interpreted.</p> <p>sText: the content to be pasted into the editor's content at the current cursor location. Any editor content that is selected when pasteText is executed is replaced.</p> <p>For example:</p> <pre>eWebEditPro.Editor1.pasteText("Hello World!");</pre>
setBodyHTML()	This method does not exist. To load HTML content into the editor, use the setDocument() method.

Method	Description
setHeadHTML()	<p>Sets the <HEAD> through </HEAD> portion of the document header.</p> <p>Syntax</p> <pre>eWebEditPro.Editor1.setHeadHTML(strReplacementHead)</pre> <p>Parameter</p> <p>strReplacementHead - The HTML <HEAD>...</HEAD> replacement string.</p> <p>Remarks</p> <hr/> <p>Do not add styles using this method. They are not supported, and the header will reflect incorrect information.</p> <hr/> <p>This feature replaces all header information. If the new header information includes styles, the style information will appear in the HEAD tag area, but will not remove, replace or add any style information.</p> <p>Example</p> <p>This replaces the header with just a TITLE element:</p> <pre>eWebEditPro.Editor1.setHeadHTML "<HEAD<TITLE>New Header</TITLE></HEAD>"</pre>
setDocument()	<p>This replaces the entire document, including all tags outside of the body tag and style information, with the specified document. Any previous document is completely lost.</p> <p>Parameter</p> <p>strDoc - String - The HTML document to place into the editor. This must be a complete and valid document that contains the doctype, html, head, and any other tags required for correct display of the document.</p> <p>Return Value:</p> <p>Nothing</p> <p>Example 1:</p> <pre>function SetFullDocument() { var objEdit = eWebEditPro.MyContent1; objEdit.setDocument(DocHTML.value); }</pre> <p>Example 2:</p> <pre><input type="button" value="Set Document" onClick="eWebEditPro.MyContent1.setDocument(DocHTML.value)"></pre>

Method	Description
setProperty (name of property, value)	<p>Writes to the ActiveX control property.</p> <hr/> <p>Note: This property is intended for environments such as Netscape, which do not directly support properties.</p> <hr/>

ActiveX Style Sheet Methods

Use these methods to manage style sheets. For more information, see "Style Sheets" on page 219.

Method	Description
addInlineStyle	<p>Adds an inline <STYLE> . . . </STYLE> to the document header.</p> <p>Syntax</p> <pre>strReturnValue = eWebEditPro.Editor1.addInlineStyle (strSelector, strStyle)</pre> <p>Parameters</p> <p>strReturnValue - If successful, strReturnValue is equal to strStyle. If unsuccessful, strReturnValue is an error message.</p> <p>strSelector - The tag to which the strStyle is applied. Note that the strSelector should not represent more than one tag. To apply the same style to multiple tags, add a style for each tag.</p> <p>strStyle - The CSS syntax style to apply to the strSelector tags in the content.</p> <p>Remarks</p> <ul style="list-style-type: none"> The new style sheet overrides rules for existing tags. For example, if a style sheet affects P, LI and DIV, and there is a call to addInlineStyle that "adds" a style for the P tag, the new P style overrides the existing P style, but the LI and DIV styles remain in effect. The strStyle syntax starts and ends with the style information. The function supplies the curly brackets that surround the style information. For example: <pre>strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")</pre> <p>Example</p> <p>This adds style H4, identified by the style title "UserH4," to the document header.</p> <pre>strNewStyle = eWebEditPro.Editor1.addInlineStyle("H4", "font-size:22pt; margin:15px; color:blue; font-family: ""Century Gothic""", "UserH4")</pre> <p>As a result, the header HTML now has this extra content.</p> <pre><STYLE title=UserH4>H4 { COLOR: blue; FONT-FAMILY: "Century Gothic"; FONT-SIZE: 22pt; MARGIN: 15px</pre>

Method	Description
addLinkedStyleSheet	<p>Adds a linked style sheet reference to the document header.</p> <p>Syntax</p> <pre>strReturnValue = eWebEditPro.Editor1.addLinkedStyleSheet(strURL)</pre> <p>Parameters</p> <p>strReturnValue - If successful, strReturnValue is equal to strURL. If unsuccessful, strReturnValue is blank.</p> <p>strURL - The URL of the style sheet to link to.</p> <p>Remarks</p> <ul style="list-style-type: none"> The new style sheet overrides rules for existing tags. For example, if a style sheet affects P, LI and DIV, and there is a call to addLinkedStyle that "adds" a style for the P tag, the new P style overrides the existing P style, but the LI and DIV styles remain in effect. <p>Example</p> <pre>strMyStyleReturn = eWebEditPro.Editor1.addLinkedStyleSheet("http://www.ourcompany.com/styles/mystyles.css")</pre> <p>As a result, the header HTML now has this extra content.</p> <pre><LINK href="http://www.ourcompany.com/styles/mystyles.css" rel=stylesheet title= http://www.ourcompany.com/styles/mystyles.css></pre>
disableStyleSheet	<p>Enables or disables a linked or inline style sheet as identified by its title.</p> <p>Syntax</p> <pre>eWebEditPro.Editor1.disableStyleSheet (strTitle, bDisabled As Boolean)</pre> <p>Parameters</p> <p>strTitle - A unique identifier that represents this style sheet. For an inline style sheet, the title is the tag that the style affects; for a linked style sheet, the title is the style sheet's URL.</p> <p>bDisabled - Boolean: True: disable the style sheet; False: enable the style sheet</p> <p>Example</p> <p>Assume that you added an inline style sheet.</p> <pre>strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")</pre> <p>This code disables that style sheet.</p> <pre>eWebEditPro.Editor1.disableStyleSheet "P", True</pre> <p>This code re-enables it.</p> <pre>eWebEditPro.Editor1.disableStyleSheet "P", False</pre>
disableAllStyleSheets	<p>Enables or disables all style sheets for an editor.</p> <p>Syntax</p> <pre>eWebEditPro.Editor1.disableAllStyleSheets()</pre>

Method	Description
BodyStyle	<p>Sets/gets the document's body style.</p> <p>BodyStyle adds an inline style to the document header. It does <i>not</i> add attributes to the Body tag.</p> <p>Syntax</p> <pre>eWebEditPro.Editor1.bodyStyle = strCssText</pre> <p>Parameters Set</p> <p>New_BodyStyle - The CSS style (without curly braces) for the new body style.</p> <p>Parameters Get</p> <p>(return value) - The CSS of the current body style.</p> <p>Example</p> <p>The following creates an inline body style that sets the document font to red Arial.</p> <pre>eWebEditPro.Editor1.bodyStyle = "color:red;font-family:Arial"</pre> <p>strBodyStyle now looks like this.</p> <pre>BODY { COLOR: red; FONT-FAMILY: Arial }</pre>

Method	Description
	<p>PopulateTagsWithStyles</p> <p>Applies the current, active styles to the content's tags.</p> <p>Syntax</p> <pre>bResult = eWebEditPro.Editor1.populateTagsWithStyles</pre> <p>Parameters</p> <p>bResult - Boolean True: Success; False: Failure</p> <p>Remarks</p> <p>Defaults</p> <p>This function adds some harmless default values: note "BOTTOM" and "FILTER" in the example below.</p> <p>Precedence</p> <p>When rendering content, styles embedded in content tags take precedence over header style tags.</p> <p>Example</p> <p>Given the style sheet added inline:</p> <pre>strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")</pre> <p>Where the resulting style is:</p> <pre><STYLE disabled title=P>P { FONT-FAMILY: Arial }</pre> <p>And content represented by this HTML:</p> <pre><P>Sentence one</P> <P>Sentence two</P> <P>&nbsp;</P></pre> <p>Calling PopulateTagsWithStyles yields:</p> <pre><P style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: Arial">Sentence one</P> <P style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: Arial">Sentence two</P> <P style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: Arial">&nbsp;</P></pre>

Method	Description
ClearStylesFromTags	<p>Removes the style attribute from all tags in the document.</p> <p>Syntax</p> <pre>eWebEditPro.Editor1.clearStylesFromTags</pre> <p>Parameters</p> <p>none</p> <p>Example</p> <p>Given the style sheet added inline and the call to PopulateTagsWithStyles:</p> <pre>Dim bResult As Boolean strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage.css") bResult = eWebEditPro1.PopulateTagsWithStyles</pre> <p>The resulting HTML looks like this.</p> <pre><H1 style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: 'Arial'; FONT-SIZE: 11pt; MARGIN: 0in">This text is styled by testpage3.css</H1> <H2 style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: 'Arial'; FONT-SIZE: 10pt; MARGIN: 0in">This text is styled by testpage3.css</H2></pre> <p>Calling ClearStylesFromTags removes the styles and produces:</p> <pre><H1>This text is styled by testpage3.css</H1> <H2>This text is styled by testpage3.css</H2></pre>

Method	Description
	<p>Returns a comma-delimited list of the titles of the active styles.</p> <p>Syntax</p> <pre>strResult = eWebEditPro.Editor1.getActiveStyleSheetTitles</pre> <p>Parameters</p> <p>strResult - The comma-delimited result set</p> <p>Example</p> <p>Given this sequence of adding styles:</p> <pre>strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\" & "ektNormal.css")</pre> <pre>strResult= eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage.css")</pre> <pre>strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")</pre> <pre>strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage3.css")</pre> <p>And this disable call:</p> <pre>eWebEditPro.Editor1.disableStyleSheet App.Path & "\" & "ektNormal.css", True</pre> <p>The call:</p> <pre>strResult = eWebEditPro.Editor1.getActiveStyleSheetTitles</pre> <p>Yields the three remaining active styles, (testpage.css, P, testpage3.css):</p> <pre>[value of App.Path]\testpage.css, P, [value of App.Path]\testpage3.css</pre>

Method	Description
	<p>ShowsActiveStylesDetails</p> <p>Returns a comma-delimited list of the active style sheet titles and style information (CSS syntax text).</p> <p>If all of a style's rules were overridden but the style is still active (that is, not "disabled"), then the value of that style returns the phrase: "No Active Rules"</p> <p>Syntax</p> <pre>strStyles = eWebEditPro.Editor1.showActiveStylesDetails</pre> <p>Parameters</p> <p>(result) - The comma-delimited list of style sheet titles and their values</p> <p>Example</p> <p>Given adding these styles:</p> <pre>strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage.css")</pre> <pre>strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage3.css")</pre> <pre>strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family: ""lucida console""")</pre> <p>Where:</p> <ul style="list-style-type: none"> • testpage.css defines a style for the P tag • testpage3.css defines styles for P, H1 and H2 • The third line inserts an inline P tag style <p>So that:</p> <ul style="list-style-type: none"> • testpage.css adds a P style • testpage3.css overrides that P style and adds a style for H1 and H2 • the inline style overrides the P style yet again <p>Calling <code>eWebEditPro.Editor1.showActiveStylesDetails</code> yields the following results.</p> <pre>Stylesheet: C:\EKTRON~1\DEVELO~1\EWEBED~1\v2\Test\TestApp\testpage.css, No Active Rules</pre> <pre>Stylesheet: C:\EKTRON~1\DEVELO~1\EWEBED~1\v2\Test\TestApp\testpage3.css, cssText: H1 { FONT-FAMILY: "Arial"; FONT-SIZE: 11pt; MARGIN: 0in } H2 { FONT-FAMILY: "Arial"; FONT-SIZE: 10pt; MARGIN: 0in} Stylesheet: P, cssText: P { FONT-FAMILY: "lucida console"</pre>

ActiveX Events

Each event corresponds to a JavaScript event handler in ewebeditprodefaults.js. (See "The ewebeditprodefaults File" on page 76).

Event	Description
ondblclickelement (ByVal oElement As Variant)	<p>Double-clicking on a hyperlink, applet, object, image, or table causes this event to fire. See the ewebeditproevents.js file for an example of how to respond to this event.</p> <p>oElement is a reference to the element object. The Variant returned is an HTML element object suitable for dynamic HTML (DHTML) scripting. See a DOM reference for complete details on the element object. A few of the most useful common properties of the element object are listed below. Other properties are dependent on the type of element.</p> <ul style="list-style-type: none">● tag name - the element's tag. For example, <code>oElement.tagName + "";</code> <p>NOTE: The plus sign (+) converts the tag name to a string.</p> <ul style="list-style-type: none">● outerHTML - the entire HTML text of the element including the tag.
onexeccommand (ByVal strCmdName As String , ByVal strTextData As String , ByVal IData As Long)	<p>The ActiveX control raises the onexeccommand after a toolbar button is pressed, a toolbar drop-down menu item is selected, or a context menu (right-click menu) item is selected.</p> <p>ByVal strCmdName As String - The command that the user action executes</p> <p>ByVal strTextData As String - Text associated with the command (typically not used)</p> <p>ByVal IData As Long - Numeric data associated with the command (typically not used)</p> <p>The IData parameter does not reflect the index of the list box item. Instead, it only returns the data assigned to the item.</p> <p>If, in the processing of the command notification, you need the index of the selected item, use <code>objCommand.getPropertyInteger("CmdIndex")</code>.</p>
onfocus()	An event that fires when the editor gains the focus. onfocus is a standard DHTML event.
onblur()	An event that fires when the editor loses the focus. onblur is a standard DHTML event.

The XML Configuration Data

VisualFormat's configuration data lets you define many aspects of the editor. For example, by modifying the configuration data, you can

- enable/disable features, such as automatic spell check
- arrange toolbars
- add custom commands
- determine whether users can edit HTML source code
- manage the image selection feature

Modifying XML Configuration Data

There are two ways that you can modify XML configuration data

- *dynamically*, using JavaScript on the server, on the client, or both.
- *statically*, by editing an .xml file that stores the configuration data. The file's name is config.xml, and by default, it is installed in the `iw/ewebeditpro20` directory.

Note

If you use an XML editor to edit config.xml, a corresponding .dtd file (config.dtd) is supplied that you can use to validate config.xml. By default, the config.dtd is installed to the `iw/ewebeditpro20` directory.

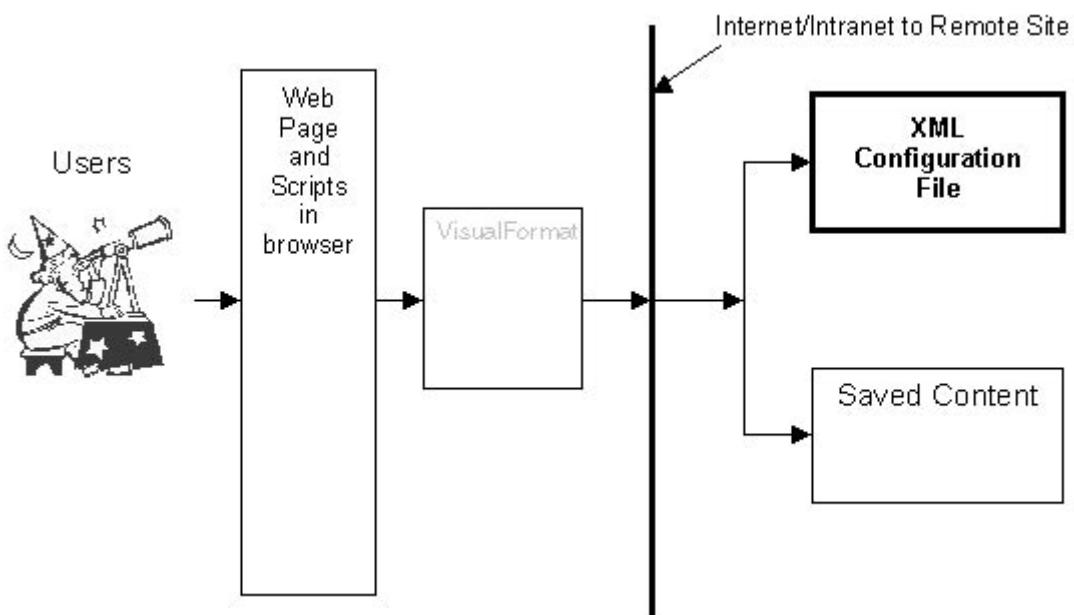
Organization of Configuration Documentation

Documentation for the configuration data consists of the following topics.

Topic	Begins on page
Step-by step instructions for defining the editor	12
Changing the Editor Dynamically	132
Managing the XML configuration data	134
Allowing users to customize the toolbar	137
Reference documentation for XML configuration data elements	141
The image selection feature	234

Managing the Configuration Data

The Site Administrator controls the configuration data and specifies which configuration data to use. Users cannot edit the configuration data.



Editing the Configuration Data

To implement a standard configuration of VisualFormat, leave the configuration data as is. If you want to modify the standard configuration data, you have two choices:

- change the configuration data dynamically (see “Dynamically Changing the Editor” on page 37.)
- edit the config.xml file using your favorite text editor or a specialized XML editor (continue reading this section)

If you edit the config.xml file, be very careful to adhere to the format. For example, if you accidentally delete a less than character (<), your edits are not applied.

XML is case-sensitive. Therefore, keep all element names (for example, <command>) and attribute names (for example, name) lower case.

If you use an XML editor to edit config.xml, Ektron supplies a corresponding .dtd file (config.dtd) that you can use to validate config.xml. By default, the config.dtd is installed to the ewebeditpro2 directory.

If you want to insert HTML as a stream into the config.xml file (as opposed to as a file specification), delete the encoding attribute information (`encoding=`) at the top of the file.

Providing Configuration Files for User Groups

Since the file is designated at run time, you can use scripting to determine which XML configuration data the user loads. The following are a few options for implementing separate configuration files for different user groups in your organization.

In this example, you create a configuration file named admin.xml.

- Set up a series of web pages for each group to log into. Each page specifies which XML configuration data to use.

For example, you could change the XML configuration data in the HTML page that launches the editor using this line.

```
<script language="JavaScript1.2">
<!--
eWebEditPro.parameters.config="administration.xml"
eWebEditPro.create("MyContent2", 700, 250);
//-->
</script>
```

(See Also: “Appendix A: Naming the VisualFormat Editor” on page 300.)

You can set up and reference different file names or different file locations. Using different file names is probably easier if you are starting with the sample files provided with VisualFormat.

-
- Use the user's login name to determine which XML configuration data to use. In an ASP or ColdFusion environment, you can use the login name as a search key in a database to retrieve the configuration data that the user should access.
 - Use the login name as the XML file name. You can keep all configuration files in one location and build the xml file name using the login name.

```
strCfgFile = "http://www.ektron.com/configs/" + Login.value + ".xml  
ewebeditpro1.Config = strCfgFile
```

This is similar to a user's profile that is set up when someone logs into an operating system.

If the user does not have a profile, the user gets the editor's default functionality.

Changing the Configuration Data's Location

The location of the XML configuration data is specified in the ewebeditprodefaults.js file, which is located in the folder to which you installed VisualFormat.

The configuration data's location is specified in the this.config = attribute. To change the location of the XML configuration data on the client, edit this line.

Allowing Users to Customize the VisualFormat Toolbar

The `allowCustomize` attribute of the `interface` element is part of the XML configuration data. Possible values are "**true**" and "**false**".

```
<interface name="beta" allowCustomize="true">
```

Modify this attribute to let users customize VisualFormat. If the attribute is set to "**true**", users can

- create a new menu
- remove an existing menu
- add commands to a menu
- remove commands from a menu
- rearrange the commands on a menu

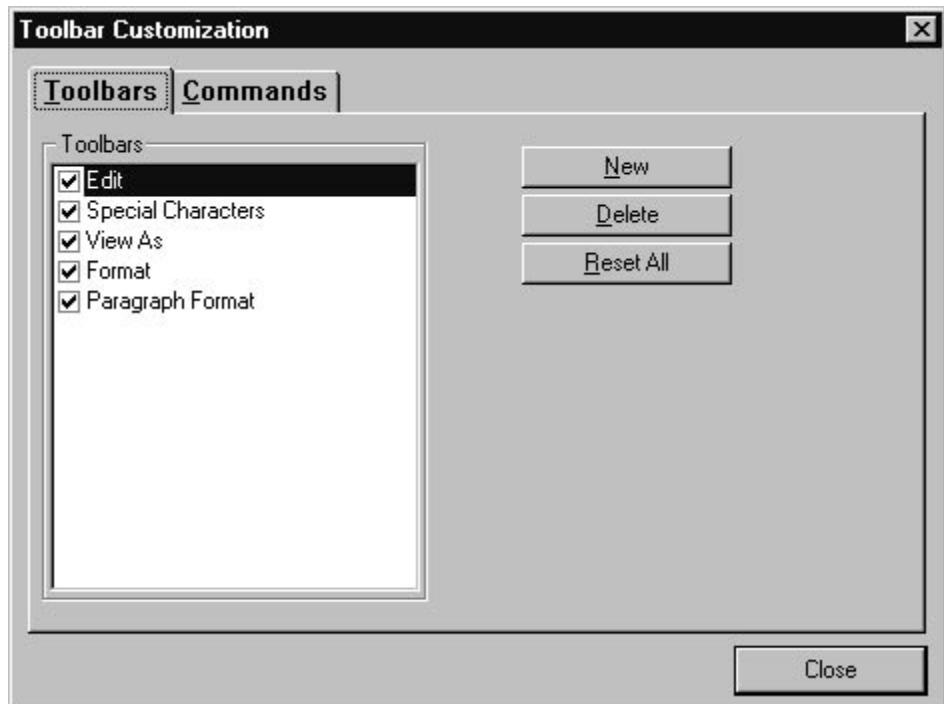
Note

Users can only add commands defined in the XML configuration data to a menu.

If `allowCustomize` is set to "**false**", the **Customize** option does not appear on the user's customization menu (the menu that the user invokes the customize the toolbar).

Allowing User Customization

The user places the cursor on a toolbar and right clicks the mouse to invoke the customize dialog box (illustrated below).



After the user customizes menus and presses **Close**, the customization files are saved in the client PC's temporary folder. The file's names (prior to the file extensions) match the value assigned to the `interface name` attribute in the configuration data.

The next time the user opens that page, the customized toolbar appears. From this point forward, any changes that you make to the interface section of the configuration data on the server are not used on the user's computer.

If you want to apply changes to the interface section of the configuration data to all users, see "Overriding User Customization" on page 139.

Preventing User Customization

If you set `allowCustomize` to "**false**", users cannot permanently customize their toolbars. The system uses the default toolbar and menu specifications defined in the XML configuration data.

NOTE

If you set `allowCustomize` to **false**, the user still sees the customize option, and customization procedure acts the same. However, the customization is only saved while the user remains on the page. Once the user leaves the page, the customization is lost.

Overriding User Customization

You might allow users to customize VisualFormat, but later need to implement a global change to the editor. For example, you may decide that users cannot edit HTML code.

To override all user customization, follow these steps.

1. Make the necessary changes in the XML configuration data on the server.
2. Change the value of the `interface name` attribute. For example, if the attribute's value is **beta**, you could change it to **beta1**.

WARNING!

If you override user customization, users lose all changes made to **eWebEditPro** toolbars and menu configurations. If the users preferred those customizations, they must redo them.

To understand how changing the `interface name` attribute of the XML configuration data updates all user configurations, read "How VisualFormat Determines Which Configuration Data to Use" on page 140.

How VisualFormat Determines Which Configuration Data to Use

When a user launches VisualFormat, the following events occur.

1. The browser reads the XML configuration data on the server to determine which data to use. The file name is the value of the `interface name` attribute.
2. The browser looks for a customization file with that name in the temporary folder on the user's computer. If it finds one, that XML configuration data determines which toolbars to display.
If the browser does not find a customization file on the user's computer, it defaults to the interface section of the XML configuration data on the server to determine which toolbars to display.

Overview of XML Configuration Data

The following illustration is an abbreviated version of the file that lets you edit the XML configuration data, config.xml. The major components highlighted in red. Following the illustration is a table that summarizes the major components in the order in which they appear in the file.

NOTE

You can also edit XML configuration data dynamically. For information, see “Dynamically Changing the Editor” on page 37.

NOTE

If you use an XML editor to edit config.xml, Ektron supplies a corresponding .dtd file (config.dtd) that you can use to validate config.xml. By default, the config.dtd is installed to the ewebeditpro2 directory.

You can review the illustration and table for an overview of these components before proceeding to subsequent sections, which provide more detail about each component.

```

xml declaration <?xml version="1.0" encoding="iso-8859-1"?>
root element<config product="eWebEditPro">

<interface name="...."
<menu name="editbar" .....
<button command="cmddelete"/>
</menu>
</interface>
<features>
<external enabled="true">

<table enabled="true">

<viewas enabled=".....">
<clean .....>
<edithtml enabled="...."/>

<standard autoclean="true" ...>
<style .....>

<command name= ....>

<spellcheck enabled="true">
<mediafiles>
</features>
```

Section	Function	Example	For more info, see page
xml Declaration	Identifies the file as an xml file.	<?xml version="1.0" encoding="iso-8859-1"?>	
Root Element	Identifies the root element of the file.	<config product="eWebEditPro">	146
Interface	Lets you define the user interface.	<interface name="beta4" allowCustomize="false">	137, 146
Menu	Defines a toolbar or pulldown menu.	<menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="false"> <caption LocaleRef="btnMainCap">Edit </caption>	164
Button	Defines a toolbar button or a menu item.	<button command="cmdcut"/>	149

Section	Function	Example	For more info, see page
Features	Defines standard and custom commands.	<features>	146
External	Defines external client functionality.	<external enabled="true">	193
Table	Allows users to create tables.	<table enabled="true">	186
ViewAs	Determines whether users can view HTML source code.	<viewas enabled="true" publish="xhtml" mode="whole">	195
Clean	Ensures that content is readable and concise HTML.	<clean cr="cr" lf="lf"/>	197
editHTML	Determines whether users can edit HTML source code.	<edithtml enabled="false"/>	196
Standard	Defines standard editing commands and options.	<standard autoclean="true" publish="xhtml">	168
Style	Applies a style sheet.	<style publishstyles="true" href="[eWebEditProPath]/ektnormal.css"/>	219
Command	Defines a standard editor action, such as cutting text. Can also define a custom action.	<command name="cmdcut" enabled="true"> <image key="Cut" /> <caption localeRef="btnTxtCut">Cut</caption> <toolTipText localeRef="btnTxtCut">Cut</toolTipText> </command>	171
Spellcheck	Controls the operation of spell checking.	<spellcheck enabled="true">	204
Mediafiles	Controls the selection and upload of media (for example, images)	<mediafiles> <command name="cmdmfumedia" style="icon" visible="true"> </mediafiles>	234

Elements that Determine the User Interface

The XML configuration data contains several elements that let you define the VisualFormat toolbar. For example, the button element lets you define the image that appears on a toolbar button, and the command that is executed when the user presses the button.

NOTE

You can also edit XML configuration data dynamically. For information, see "Dynamically Changing the Editor" on page 37.

Below is an alphabetical list of each element in the standard feature. A full description appears later in this section.

Element	Description	For more information, see page
bar	Separates a group of commands from other commands on a menu.	148
button	a toolbar button or menu item.	149
caption	The text that describes a menu bar or toolbar button.	150
command	A standard or custom editor action, such as copying text.	152
cmd	An abbreviated version of <command>.	154
config	The single root element that signifies that this configuration belongs to the VisualFormat Release.	155
features	One of the two major sections of the XML configuration data. Defines all standard and custom commands, and publishing options.	156
image	An image to display on a button.	157
interface	One of two major sections of the XML configuration data. Defines toolbars, menus, dialogs, and other interface items.	159
listchoice	Each item on a list.	161
mediafiles	Controls media file selection.	234

Element	Description	For more information, see page
menu	A toolbar or pulldown menu.	164
popup	A menu that is launched by pressing a toolbar button.	166
selections	A group of listchoice items.	166
space	A separator between toolbar buttons or popup menus.	167
standard	Defines standard editing commands and options.	168
toolTipText	The text that appears when the cursor hovers over a toolbar button.	169

Hierarchy of Elements

To understand how elements of the XML configuration data work together, review the following outline. The outline does not include the attributes that let you define each element. The attributes are defined with the description of each element, later in this section.

```

Config
  Interface
    Menu
      Button (uses command)
      Bar
      Space
      Caption
    Popup
      Button (uses command)
      Bar
  Features
    Feature name
      Command or cmd
      Caption
      Tool Tip Text
      Image
      Selections
      Listchoice

```

The Config Element

Config is the root element that contains all information about the elements of VisualFormat that you are defining. All other elements are defined within the config element.

Therefore, you create different XML configuration data for every unique set of functions that you are implementing. For example, if one user group can view source code, while another group cannot, you would create two sets of XML configuration data.

Users can customize VisualFormat toolbars. See “Allowing Users to Customize the VisualFormat Toolbar” on page 137 for details.

The Interface Element

Use the interface section of the XML configuration data to define the user interface. Within the interface element, you can modify

- which toolbar buttons are available to the user
- the sequence of toolbar buttons
- space between toolbar options

NOTE A toolbar button typically executes a command. Commands are defined in the Features element.

Buttons not Assigned to Menus

By default, some commands are not assigned to any standard menu. However, the user can place any enabled command on a menu.

The Features Element

Use the features section of the XML configuration data to define the commands that are assigned to buttons and menus in the interface section. You can

- delete standard commands
- add custom commands
- modify the images and text that appear with the command on a toolbar button or menu

-
- if the command's style is list, enter the listchoice items on the list
 - set feature options, such as enabling publishing options

Attribute Types

Each element has one or more attributes that let you tailor its function to your unique needs. Each attribute is one of the three types listed below.

The attributes are actually always strings, but the editor expects their values to be one of the types listed below.

Boolean

These are the valid string values for Boolean attributes.

Positive	Negative
yes	no
true	false
1	0
ok	[unknown]

Integer

An attribute that is expected to contain numeric values is interpreted as an integer. If an integer value contains alpha characters, it is converted to 0.

String

An attribute interpreted as this type uses the text given without interpretation. All characters are converted to lower case unless the text is defined as a path.

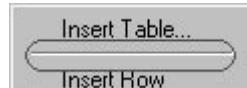
Element Definitions

bar

Places a



- vertical bar on a toolbar or



- horizontal bar on a popup menu

The bar separates a group of commands from other commands on a menu.

See also “Adding a Separator Bar Between Two Toolbar Buttons” on page 20.

NOTE Unlike the other commands, the bar and space elements are not defined in the XML configuration data. You cannot modify their appearance.

Element Hierarchy

```
<config>
  <interface>
    <menu>
      <bar>
```

```
<config>
  <interface>
    <popup>
      <bar>
```

Attributes

Name	Attribute Type	Default	Description
None			There are no attributes to the bar element.

Example

```
<menu name="editbar">
    <caption visible="false" localeRef="btnMainCap">Edit</caption>
    <button command="cmdcut" />
    <button command="cmdcopy" />
    <bar/>
    <button command="cmdpaste" />
</menu>
```

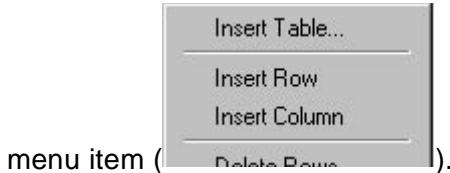
The result of this code would look like this.



button



Defines either a toolbar button () or a



menu item ().

The button element has a command attribute that identifies a command to execute when the user selects a toolbar button or a menu item. The value assigned to the button's command attribute must be defined within a command element. If not, the command is not added to the icon bar or menu item.

The order in which button elements are entered on a menu or popup menu command determines the order in which menu items appear on icon bars or menus.

See also “Adding a Standard Toolbar Button” on page 17 and “Removing a Toolbar Button” on page 18.

Element Hierarchy

```
<config>
    <interface>
        <menu>
            <button>

<config>
    <interface>
        <popup>
```

```
<button>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the command enabled? If false, the button is grayed.
popup	String	""	Defines a Popup menu to display when the button is selected. (See "popup" on page 166.) If a popup is defined, the command name is not sent to the client.
command	String (must be defined as a command element)	""	The command to execute when the user clicks the button.

Example

```
<menu name="samplebar">
  <caption visible="false" localeRef="btnMainCap">Sample</caption>
  <button command="cmdcut" />
  <button command="myselections" popup="myPopup" />
</menu>
```

caption

Provides the text that describes a menu bar or toolbar button in the user interface.

If a caption is assigned to a menu, the caption text only appears



when the menu bar is floating.

If a caption is assigned to a button, the caption text appears on the toolbar with the icon if you are displaying button caption text.



NOTE

The `textAlignment` attribute of the `menu` element determines the alignment of text within a button.

See also “Creating or Editing the Menu Caption” on page 16 and “Displaying Button Caption Text” on page 21.

Element Hierarchy

```
<config>
  <features>
    <external>
      <command>
        <caption>

<config>
  <interface>
    <menu>
      <caption>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the element enabled?
localeref	String	""	Localization identifier. This value translates the included phrase into the local language. See also “Translating Button Captions and Tool Tips to a Foreign Language” on page 23.
visible	Boolean	Yes	Is the caption visible by default?

Examples

Button

```
<command name="cmdviewashtml" style="icon" visible="true">
    <caption localeRef="btnTxtVAHtml">View As HTML </caption>
```

Menu

```
<menu name="editbar" newRow="false" showButtonsCaptions="true"
    textAlign="bottom" wrap="false">
    <caption visible="true" localeRef="btnMainCap">Edit</caption>
</menu>
```

command

Defines a standard editor action, such as copying text. Before commands can be used by a toolbar button or menu item, they must be defined and enabled.

See also “Commands” on page 171 and “cmd” on page 154.

Element Hierarchy

```
<config>
    <features>
        <external>
            <command>
```

and

```
<config>
    <features>
        <standard>
            <command>
```

A command definition does not need to be directly under a feature element. But it must be contained somewhere within a feature hierarchy.

Attributes

Name	Attribute Type	Default	Description
name	String	""	The command's name. The name must be unique.

Name	Attribute Type	Default	Description
style	String	default	<p>The style of the command when it appears on an icon bar or menu. The command can be one of these styles.</p> <ul style="list-style-type: none"> ● "icon" ● "toggle" ● "listbox" ● "edit" <p>For more information, see "Command Styles" on page 154.</p>
enabled	Boolean	Yes	Is the command enabled? If false, the command is not created.
maxwidth	Number	10	The maximum number of characters wide to make a command. This attribute only applies when the command is a list box or edit box.
image	String (key and/or src)	""	<p>The image that appears if the command is assigned to a button.</p> <p>See also "Button Images" on page 179.</p>
caption	String	""	<p>Provides the text that describes a menu bar or toolbar button in the user interface.</p> <p>See also "caption" on page 150.</p>
toolTipText	String	""	<p>Defines the tool tip text that pops up when the cursor hovers over an icon.</p> <p>See also "toolTipText" on page 169.</p>
ref	String	""	<p>Can replace caption or toolTipText, or both.</p> <p>Enter a code from the locale file to define a command's caption or ToolTiptext or both.</p>
visible	Boolean	True	<p>Is the command visible when first created?</p> <p>If set to false, the command is created but is not displayed by default.</p>
selections	String	""	<p>Defines a list of items within a listchoice command.</p> <p>See also "selections" on page 166.</p>

Command Styles

The command can be one of these styles.

Style	Values that indicate this style in style attribute	Description
"icon"	default, icon, or unknown	A toolbar button that is drawn as a rectangle normally containing an image. The button can contain both an icon and a caption, just the icon, or just the caption.
"toggle"	toggle	A button that maintains a pressed or checked state. If shown in a list box, it displays with a check. If shown on a toolbar, it is drawn as an 'icon' style command but is pressed in when checked. Clicking on the item toggles its state between check and unchecked or pressed in and popped out. If drawn on a toolbar, it is drawn using the same options as the 'icon' style.
"listbox"	listbox, list	This creates a command button that is displayed as a drop down list box. The items for this listbox are defined in the selections element which is contained within the command element defining the command. (See "listchoice" on page 161.)
"edit"	edit, text	This creates a command button that allows the user to enter text. For each character typed into the edit area, the command is sent with the current text as the command's parameter.

Example

```
<command name="cmdviewashtml" style="icon" visible="true">
  <caption localeRef="btnTxtVAHtml">View As HTML </caption>
  <toolTipText localeRef="btnVAHtml">View As HTML</toolTipText>
</command>
```

cmd

An abbreviated version of command, created to reduce the time required to load XML configuration data. Note that `<cmd>` has fewer attributes than `<command>`.

The `<command>` element is still available. You must use it for more complex commands, such as drop-down lists.

Element Hierarchy

```
<config>
  <features>
    <standard> or any other feature
    <cmd>
```

Attributes

Name	Attribute Type	Default	Description
name	String	""	The command's name. The name must be unique.
style	String		<p>The style of the command when it appears on an icon bar or menu.</p> <ul style="list-style-type: none">• "icon"• "toggle"• "listbox"• "edit" <p>For more information, see "Command Styles" on page 154.</p>
enabled	Boolean	Yes	Is the command enabled? If false, the command is not created.
key	String		<p>The image that appears if the command is assigned to a button.</p> <p>See also "Button Images" on page 179.</p>
src	String		<p>The image that appears if the command is assigned to a button.</p> <p>See also "Button Images" on page 179.</p>
ref	String	“ “	<p>Replaces caption and toolTipText.</p> <p>Enter a code from the locale file to define a command's caption and ToolTiptext.</p>

config

The single root element that signifies that this configuration belongs to the eWebEditPro Release 2. This entry must exist before the configuration information is processed.

See also "The Config Element" on page 146.

Element Hierarchy

```
<config>
```

Attributes

Name	Attribute Type	Default	Description
product	String	" "	The XML configuration data's target product. This attribute's value must be eWebEditPro for processing to continue.
version	Integer	0	The product release for which this XML configuration data is targeted. The value must be 2 or greater for processing to continue.
revision	Integer	0	The revision of the target product.

Example

```
<config product="eWebEditPro" version="2" revision="1">
```

features

One of the two major sections of the XML configuration data, the features section defines all standard and custom commands, and publishing options.

All features loaded into the product must be defined within this element. Any feature defined outside is ignored.

See also “The Features Element” on page 146.

Element Hierarchy

```
<config>
  <features>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If set to false, all features are disabled and no commands are created in the interface. If false, the client or script must use the Menus object to create any necessary commands. Note that the standard feature cannot be disabled. For this feature only, the enabled attribute is ignored.

Example

```
<features enabled="true">
```

image

Specifies an image to display for a command. If the command style is toolbar button, the image appears on the popup or menu button.

See also “Button Images” on page 179, and “Changing the Image that Appears on a Toolbar Button” on page 20.

Element Hierarchy

```
<config>
  <features>
    <external>
      <command>
        <image>
```

Attributes

Name	Attribute Type	Default	Description
key	String	default	The name of the internal image to display. If an image has both a key and a src value, the src value overrides the key value.
src	String	" "	The location of an external image. Since this is seen as a path, the character case is preserved.

Example

Using only a Key Attribute

```
<command name="cmdCut" style="icon" visible="true">
  <image key="Cut" />
  <caption localeRef="btnTxtCut">Cut</caption>
  <toolTipText localeRef="btnTxtCut">Cut a selection</toolTipText>
</command>
```

Using Both a Key and an Src Attribute

```
<command name="mysaveaspif" style="icon" visible="true">
  <!--The src attribute takes precedence over the key attribute -->
  <image key="spellcheck"
    src="http://us.al.yimg.com/us.yimg.com/i/ww/gift1.gif"/>
  <toolTipText localeRef="btntsapf">Save as PIF</toolTipText>
</command>
```

interface

The section of the XML configuration data that defines toolbars, menus, dialogs, and other interface items. Interface items defined outside this section are ignored.

See also “The Interface Element” on page 146.

Element Hierarchy

```
<config>
  <interface>
```

Attributes

Name	Attribute Type	Default	Description
allowCustomize	Boolean	Yes	<p>Determines whether users can customize their interface from the one defined in the XML configuration data.</p> <p>If True, the user can modify toolbars. The customization is saved on the local system under the name given in the name attribute.</p> <p>If you set this value to False, the editor ignores any customization that the user saves. In this case, the default interface is used.</p> <p><i>See Also:</i> "Allowing Users to Customize the VisualFormat Toolbar" on page 137.</p>
enabled	Boolean	Yes	<p>Determines whether the interfaces defined here are enabled. If set to False, it is the responsibility of the client or script to use the Menus object to create the interface.</p> <p><i>See Also:</i> "Dynamically Changing the Editor" on page 37.</p>
name	String	Default	<p>The name of the interface. When a user customizes their interface, this name identifies the changes.</p> <p>One method of resetting an interface to allow for customization, but ignore previous customization, is to change the name. This will ignore a saved configuration and use the one defined in the XML configuration data.</p> <p><i>See Also:</i> "Allowing Users to Customize the VisualFormat Toolbar" on page 137.</p>

Name	Attribute Type	Default	Description
visible	Boolean	True	<p>Controls whether the toolbar is visible. If set to false, the interface is created but does not appear. However, the context menu appears if a user right clicks the mouse.</p> <p>If set to false, the toolbar can only be displayed by programmatically by calling <code>ShowAllMenus()</code> in the Menus interface, using a script like this:</p> <pre>eWebEditPro.MyContent1.Menus().ShowAllMenus();</pre> <p>(See Also: "Method ShowAllMenus()" on page 60.)</p> <p>For example, you set this attribute to false because the editor is the second one on a page. The XML data would look like this:</p> <pre><interface name="standard1" allowCustomize="true" visible="false"></pre> <p>But, if the user's focus shifts to the second editor, you want to display its toolbar. At that point, you display the toolbar using this script:</p> <pre>eWebEditPro.MyContent2.Menus().ShowAllMenus();</pre>
context	Boolean	True	<p>Controls whether the context menu is visible. If set to true, a menu appears when the user right clicks the mouse with choices that are unique to the current situation (or context).</p> <p>For example, if you are editing text and right clicks the mouse, the context menu displays common editing commands, such as cut and copy text. If you are editing a table, the context menu displays commands relevant to that activity, such as insert row and insert column.</p> <p>If set to false, the context menu does not appear.</p>

Example

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro" version="2" revision="1">
<interface name="myinterface" allowCustomize="true">
```

listchoice

Defines an individual choice in a list box command item.



Use this element to define attributes for each item in a list. The listchoice command's `style` attribute must be set to **List** or **Listbox**.

See also “Determining which Fonts, Font Sizes, and Headings are Available” on page 33.

Element Hierarchy

```
<config>
  <features>
    <external>
      <command>
        <selections>
          <listchoice>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the item enabled? If not, it is excluded from the list.
command	String	" "	The command to send in place of the command that contains the list. If not specified or empty, the listchoice command is sent, and the selection's index or assigned data is sent as parameters.
data	Integer	0	This value is assigned to the item selection. It is sent with the command as a parameter. If the value assigned is zero (0), the index of the selection is sent.
localeRef	String	" "	Used to translate the #text attribute of the element.

Name	Attribute Type	Default	Description
#text	String	" "	<p>The command checks the body of the listchoice element to see if it includes text.</p> <ul style="list-style-type: none"> • If it does, that text is sent as the attribute. • If the element does not include text, the command's caption attribute is sent as the attribute. <p>Also, this text is the selection item text. In other words, it is the list of options that the user sees on the drop-down menu.</p> <p>This applies whether the body of the listchoice element includes text or if the command's caption attribute is used.</p>

Example

Here is the list that creates the font choice menu shown above. Note that the items appear in the listbox in the order in which they are entered into the command.

NOTE

The top item in the list is the default value, unless the list is a font name, font size, or header style list. In that case, the currently selected item is the default value.

```
<command name="cmdfontname" style="list" visible="true" maxwidth="12">
    <image key="fontname" />
    <caption localeRef="btntxtfntnm">Times New Roman, Arial</caption>
    <toolTipText localeRef="bttnfntnm">Font</toolTipText>
        <selections name="headinglist" enabled="true" sorted="true">
            <listchoice>Arial, Helvetica</listchoice>
            <listchoice>Comic Sans MS</listchoice>
            <listchoice>Courier New, Courier</listchoice>
            <listchoice>Symbol</listchoice>
            <listchoice>Times New Roman, Times</listchoice>
            <listchoice>Verdana, Helvetica</listchoice>
        </selections>
    </command>
```

Using the Selections Element

You can use the selections element to define a group of items in a list. This can be helpful when you want to enable or disable a group of elements from one line of the XML configuration data.

Parameters to the Listchoice Command

When a listchoice command is executed, three parameter values are sent along with the command. Note that all commands can include a name and a text parameter.

Parameter	How Value Determined
name	The command checks to see if a command attribute is assigned to the listchoice element. <ul style="list-style-type: none">• If a command attribute is assigned to the element, the system sends that command.• If a command attribute is not assigned to the element, the system sends the higher level command to which the listchoice command is assigned.
text	The command checks the body of the listchoice element to see if it includes text. <ul style="list-style-type: none">• If it does, that text is sent as the parameter.• If the item does not include text, the defined command's caption attribute is sent as the parameter.
data	The data value assigned to the item selection. If the value assigned is zero (0) (the default value), the index of the selection is sent.

Assigning Command Attributes to Listchoice Elements

If you wish to send a list item as a command rather than parameter data, place a command attribute in each listchoice element.

Commands assigned as attributes to listchoice elements do not need to be defined as other commands are (that is, under the commands section of the XML configuration data). If a command is defined under the commands section of the XML configuration data, information about that command (such as the caption) is used with the selection.

Not Assigning Command Attributes to Listchoice Elements

If no command attributes are assigned to a listchoice element, the command that contains the list is sent instead, and the index or data of the selection is sent as a parameter.

Example

```
<command name="mylist" style="list" visible="true">
  <image key="Spelling"/>
  <caption localeRef="btnTxtsapf">Dropdown List of Commands</caption>
  <toolTipText localeRef="btntsapf">Dropdown List of Commands </toolTipText>
  <selections name="testlist" enabled="true" sorted="true">
    <listchoice command="cmdcopy" data="50" enabled="true"></listchoice>
    <listchoice command="cmdcut" data="8"></listchoice>
    <listchoice command="cmdpaste" data="107" localeRef="cmpaste">Just Paste</listchoice>
    <listchoice command="mynotify" data="42" localeRef="mynotify">Undefined Command </listchoice>
    <!--These selections generate the larger command name "mylist" -->
    <listchoice data="99" enabled="false">Do Not Show This</listchoice>
    <listchoice data="999" enabled="true">Selection B</listchoice>
  </selections>
</command>
```

mediafiles

See “The Mediafiles Feature of the XML Configuration Data” on page 256.

menu

Defines a toolbar or pulldown menu.

A menu is the interface between the user and the commands.

See also “Creating a Popup Menu” on page 31 and “Determining Which Menus Appear on the Toolbar” on page 14.

Element Hierarchy

```
<config>
  <interface>
    <menu>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	Is the menu enabled? A false value prevents the creation of the menu.
name	String	" "	The menu name. You refer to a menu by its name. The name must be unique.

Name	Attribute Type	Default	Description
newrow	Boolean	Yes	If the menu style is icon bar, a yes value forces the menu to a new line on the toolbar. See also "Placing a Menu on a Row with Another Menu" on page 15.
showbuttonscaptions	Boolean	False	If true, button captions are shown. Otherwise, tool tips act as the caption. This is one of the few attributes that default to false. See also "Creating or Editing the Menu Caption" on page 16 and "Translating Button Captions and Tool Tips to a Foreign Language" on page 23.
style	String	icon	Defines the look of the menu. These are the styles. "Icon" (default) - Toolbar "Pulldown" - Drop down list "Tab" - Tab Selections "Status" - Status bar "Popup" - Context Menu (See Also: "popup" on page 166)
textalignment	String	Yes	Alignment of the text on the button. (Only used if showbuttonscaptions is set to "true" .) These are the valid values. "Top" "Left" "Right" "Bottom" "Center" The default value is Bottom . See also "Defining the Alignment of Caption Text" on page 22.
visible	Boolean	True	Is the menu visible at creation time?
wrap	Boolean	True	If true, and a toolbar, when the icons reach the right edge of the display area, they wrap to the next line. If false, the icons do not wrap to the next line. They are invisible until you move the menu bar to another line of the toolbar. See also "Determining if a Menu Should Wrap to the Next Row" on page 16.

Example

```
<menu name="editbar" newRow="true"
    showButtonsCaptions="false" textAlign="bottom">
    <caption visible="false" localeRef="btnMainCap">Edit</caption>
    <button command="cmdcut" />
    <button command="cmdcopy" />
    <button command="cmdpaste" />
</menu>
```

popup

Defines a popup menu. This menu is pre-defined for use either as a stand-alone menu that is invoked programmatically, or as a menu attached to a command button.

For more information, see “Creating a Popup Menu” on page 31.

Element Hierarchy

```
<config>
    <interface>
        <popup>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the menu enabled? If set to false, you cannot create this menu.
name	String	""	The menu name. You refer to a menu by its name. The name must be unique.

Example

```
<popup name="ViewAsPopup"
    <caption visible="0" localeRef="btnMyViewAs">View As</caption>
    <button command="cmdviewaswysiwyg" />
    <button command="cmdviewashtml" />
</popup>
```

selections

Defines a list of items within a listchoice command.

This element can be helpful when you want to enable or disable a group of elements from one place.

Element Hierarchy

```
<config>
    <features>
        <external>
            <command>
                <selections>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the list enabled? If no, then the defined list is ignored.
name	String	""	The name of the list. It must be unique.
sorted	Boolean	True	If set to <ul style="list-style-type: none">• "true", the list appears in alphabetical order.• "false", the list appears as entered in the configuration data

Example

```
<selections name="testlist" enabled="true" sorted="true">
    <listchoice command="cmddcopy" data="50" enabled="true"></listchoice>
    <listchoice command="cmdcut" data="8"></listchoice>
    <listchoice data="999" enabled="true">Send testlist Command
    </listchoice>
</selections>
```

space

Places a blank separator between toolbar buttons or popup menus. On a toolbar, a space is one half the width of a normal icon (8 pixels).

The space command makes the toolbar easier to read.

NOTE

Unlike the other commands, the bar and space elements are not defined in the XML configuration data. You cannot modify their appearance.



Buttons with a Space Command



Buttons without a Space Command

See also “Adding a Space Between Two Toolbar Buttons” on page 19.

Element Hierarchy

```
<config>
  <interface>
    <menu>
      <space>
```

```
<config>
  <interface>
    <popup>
      <space>
```

Attributes

Name	Attribute Type	Default	Description
None			There are no attributes to the space element.

Example

```
<menu name="editbar">
  <caption visible="false" localeRef="btnMainCap">Edit</caption>
  <button command="cmdcut" />
  <button command="cmdcopy" />
  <space/>
  <button command="cmdpaste" />
</menu>
```

standard

Defines standard editing commands and options.

Element Hierarchy

```
<config>
  <features>
    <standard>
```

Attributes

Name	Attribute Type	Default	Description
autoclean	Boolean	"True"	<p>Whether the editor automatically detects content created by Microsoft Office 2000 applications (e.g., Word 2000). Office 2000 content may cause problems when eWebEditPro users try to reformat it (e.g., change the font size).</p> <p>When the editor detects this content, a message asks the user whether or not to clean the HTML code. (Answering yes to the prompt is the same as selecting Clean HTML from the right-click menu.)</p> <p>"false" - Do not detect Office 2000 content.</p> <p>"true" (default) - Detect Office 2000 content.</p>
publish	String	"xhtml"	<p>The quality of the HTML code output by the editor.</p> <p>When the TextHTML ActiveX control property is read, the HTML code is processed to be saved and eventually displayed. The HTML code is typically processed beyond what appears when viewed with "View As HTML".</p> <p>"Minimal" - Eliminates invalid fonts, filters image urls, and replaces cr/lf according to the values set in those attributes.</p> <p>"Cleanhtml" - Medium level: eliminates overlapping tags, and merges font tags.</p> <p>"xhtml" - The HTML code is converted to the XHTML 1.0 standard (as defined at http://www.w3.org/TR/xhtml1/).</p>

toolTipText

Defines the tool tip text that pops up when the cursor hovers over



an icon.

Element Hierarchy

```
<config>
  <features>
    <external>
      <command>
        <toolTipText>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the element enabled?
localeRef	String	""	Used as an identifier to translate the element's #text attribute.
#text	String	""	The text in the tool tip.

Example

```
<command name="cmdLeft" style="2" visible="true">
    <image key="Left" />
    <caption localeRef="btnTxtal">Left</caption>
    <toolTipText localeRef="bttnal">Align Left</toolTipText>
</command>
```

Commands

Commands define standard editor actions, such as copying text.

Each feature contains a set of commands that the developer can make available to the user on a toolbar or a menu. For example, the standard feature contains all of the standard editing commands, such as cutting and pasting text.

Note

When customizing a menu, users can assign any enabled command to a button even if the command is not assigned to any standard button.

The format of a command is:

```
<command name="<commandname>" [enabled="<true|false>"]...>  
</command>
```

For a complete list of command element's attributes, see "command" on page 152.

Topics Covered in This Section

This section covers the following topics about commands.

- General guidelines for using commands
- List of standard commands
- Special character commands
- Custom commands

See Also: "Using JavaScript to Send Commands" on page 24

Command Guidelines

Commands follow these guidelines.

- The element name is "command" or "cmd".
- The element defines a specific command that a feature supports.
- A command's name
 - is defined in the `name` attribute of the command element
 - must follow the XML standard for names
 - must be unique within the XML configuration data
- A command's definition defines its image, style, caption, and `tooltiptext`.

Standard Commands

The following table lists the commands available within the standard feature of the XML configuration data.

NOTE You can write JavaScript to execute a standard command. For more information, see "Using JavaScript to Send Commands" on page 24.

Command Name	Function
cmdabout	Displays the About eWebEditPro screen.
cmdbold	Applies bold to selected text.
cmdbookmark	Applies a bookmark to selected text.
cmdbullets	Assigns bullets to selected text.
cmdcenter	Center justifies selected text.
cmdclean	Cleans the HTML code (see "Cleaning HTML" on page 197).
cmdcopy	Copies selected text into the copy buffer.
cmdcut	Deletes selected text and places it in the copy buffer.
cmddelete	Deletes selected text.
cmdfind	Launches the Find dialog box.

Command Name	Function
cmdfontcolor	<p>Launches the Windows palette box. When the user selects a color, it applies to the selected text.</p> <p>You can also assign a color using JavaScript. To do this, send the cmdfontcolor command to the editor.</p> <p>For example, a toolbar button opens a web page that displays the 216 web safe colors. When the user clicks a color, your page needs to execute JavaScript (shown below) that assigns the color.</p> <pre>eWebEditPro[sEditorName].ExecCommand("cmdfontcolor", strColorValue, 0);</pre> <p>In this script, strColorValue is a valid HTML color string, such as, "lightgreen" or "#C0C0C0". For more information on valid color strings, see "Describing Colors" on page 175.</p> <p>To display the standard Windows color palette, pass an empty string as the color value. Here is an example.</p> <pre>// Open standard windows color palette eWebEditPro[sEditorName].ExecCommand("cmdfontcolor", "", 0);</pre>
cmdfontcolorvalue	Lets you create a list of colors that can appear as a drop-down list on the toolbar. For more information, see "The cmdfontcolorvalue Command" on page 175.
cmdhr	Inserts a horizontal line (that is, an <hr> tag).
cmdfontname	Assigns selected font name to selected text (see "fontname" on page 187).
cmdfontsize	Assigns selected font size to selected text (see "fontsize" on page 189).
cmdheaderlevel	Assigns selected header level to selected text (see "headings" on page 190).
cmdhelp	Launches the web page specified in the <helpfile src=...> attribute of the config.xml file.
cmdhyperlink	Lets you edit a hyperlink assigned to selected text.
cmdindentleft	Decreases the indentation of selected text.
cmdindentright	Increases the indentation of selected text.
cmditalic	Applies italic to selected text.
cmdleft	Left justifies selected text.

Command Name	Function
cmdmfuinsert	<p>Initiates the insertion of a selected image or file. It is the command sent when the <code>insertMediaFile</code> function is called. (See “insertMediaFile” on page 104.)</p> <p>You only need to insert this command if the function cannot be called.</p> <p>The command uses the information already given to the <code>MediaFile</code> object. The string parameter to the command should be empty. The long parameter is 1 if the file is local, and 0 if it is remote.</p> <p>The function <code>insertMediaFileDeferred</code> (called from <code>insertMediaFile</code>) demonstrates its use.</p> <pre><code>function eWebEditProEditor_insertMediaFileDeferred(strSrcFileLocation, bLocalFile, strFileType, strTitle, nWidth, nHeight) { // Place the file information into the media file object. // This is used for the insertion of the HTML. var objMedia = this.editor.MediaFile(); objMedia.setProperty("IsLocal", bLocalFile); objMedia.setProperty("SrcFileName", strSrcFileLocation); objMedia.setProperty("FileTitle", strTitle); objMedia.setProperty("FileType", strFileType); objMedia.setProperty("ImageWidth", nWidth); objMedia.setProperty("ImageHeight", nHeight); this.editor.ExecCommand("cmdmfuinsert", "", bLocalFile); }</code></pre>
cmdmfumedia	Launches the Picture dialog box. (For more information, see “ Managing Image Selection ” on page 234.)
cmdnumbered	Applies numbers to selected text.
cmdpaste	Pastes the contents of the copy buffer.
cmdpicture	Launches the Picture dialog box.
cmdredo	Reverses the most recent undo command.
cmdright	Right justifies selected text.
cmdselstyle	Displays a drop-down list of style classes. Users can select from the list to apply a style class to selected text. (See “ Implementing Style Class Selectors ” on page 224.)
cmdstrike	Applies strikethrough to selected text.
cmdsub	Applies subscript to selected text.
cmdsups	Applies superscript to selected text.
cmdtable	Contains the functionality to create and convert tables. (For more information, see “ Table Commands ” on page 186.)
cmdunderline	Applies an underline to selected text.
cmdundo	Reverses the most recent command
cmdunlink	Removes a hyperlink assignment from selected text.

The cmdfontcolorvalue Command

The cmdfontcolorvalue command lets you create a list of colors that can appear as a drop-down list on the toolbar. The user can choose a color from the list and apply it to selected text. Below is an illustration of what the drop-down list might look like.



For each listchoice data item, enter a data value (in this case, a color) and a description.

Below is a sample of how you might set up this command. This sample is provided in the config.xml file.

```
<command name="cmdfontcolorvalue" style="list" enabled="true">
  <caption>Font Color</caption>
  <toolTipText>Font Color</toolTipText>
  <selections name="fontcolorlist" enabled="true" sorted="true">
    <listchoice command="noop">Select Color</listchoice>
    <listchoice data="#00FFFF">Aqua</listchoice>
    <listchoice data="#000000">Black</listchoice>
    <listchoice data="#0000FF">Blue</listchoice>
    <listchoice data="#FF00FF">Fuchsia</listchoice>
    <listchoice data="#808080">Gray</listchoice>
    <listchoice data="#008000">Green</listchoice>
    <listchoice data="#00FF00">Lime</listchoice>
    <listchoice data="#800000">Maroon</listchoice>
    <listchoice data="#000080">Navy</listchoice>
    <listchoice data="#808000">Olive</listchoice>
    <listchoice data="#800080">Purple</listchoice>
    <listchoice data="#FF0000">Red</listchoice>
    <listchoice data="#C0C0C0">Silver</listchoice>
    <listchoice data="#008080">Teal</listchoice>
    <listchoice data="#FFFF00">Yellow</listchoice>
    <listchoice data="#FFFFFF">White</listchoice>
  </selections>
</command>
```

Describing Colors

To describe a color, use the standard HTML hexadecimal color reference, #RRGGBB, where RR is the red value, GG is green, and BB is blue. Each segment can have a value between 00 (0) and FF (255). Black is #000000, and white is #FFFFFF.

Alternatively, you can enter the color as a decimal value. To do this, omit the leading pound sign (#) character, for example, 16711935.

Note that some browsers do not support all colors.

Assigning a Color Using JavaScript

You can send the `cmdfontcolorvalue` command using JavaScript, where the value is passed as the `IData` argument. You can also use the `cmdfontcolor` command to assign the color using a string instead of a numeric value. To learn how to do this, see “`cmdfontcolor`” on page 173.

Assigning a Text Value to the Selection

The text value between the `listchoice` tags appears in the drop down list. The text can be the color name, such as **red**, in English or another language. The text can also describe the color’s purpose, for instance, **Company Logo Blue** or **Keyword Color**.

See Also: “Creating a List Item that Generates No Command” on page 35.

Displaying a Palette of Colors

As you can with any list box, you can add a list item that executes a command. If you want to let users select a color from the Windows palette box, you could add the following item to the list:

```
<command name="cmdfontcolorvalue" style="list" enabled="true">
```

You could also use the sample code provided below to open a popup window that displays a palette of colors from which the user can choose. You need to add JavaScript to process the custom `jswebcolors` command. Notice that a `command` attribute is present rather than the `data` attribute.

```
<listchoice command="jswebcolors">Web Safe Palette</listchoice>
```

Special Character Commands

You can use the following commands to create toolbar buttons that let users insert special characters. For example, cmdchr128 inserts the Euro character (€), whose Microsoft Windows Extended Character Reference value is 128.

NOTE These characters may not appear on older browsers.

To learn how to place a command on a toolbar button, see "Adding a Standard Toolbar Button" on page 17.

Character	Command name	Image key	Description
€	cmdchr128	euro	euro
,	cmdchr130	lsquor	low left single quote
f	cmdchr131	fnof	function of, florin
"	cmdchr132	ldquor	low left double quote
...	cmdchr133	hellip	ellipsis
†	cmdchr134	dagger	dagger
‡	cmdchr135	ddagger	double dagger
%o	cmdchr137	permil	per thousand sign
Š	cmdchr138	sscaron	S caron
‘	cmdchr139	lsquo	left single angle quote
Œ	cmdchr140	oeoelig	OE ligature
Ž	cmdchr142	zzcaron	Z caron
‘	cmdchr145	lsquo	left single quote
’	cmdchr146	rsquo	right single quote
“	cmdchr147	ldquo	left double quote
”	cmdchr148	rdquo	right double quote
•	cmdchr149	bull	round filled bullet

Character	Command name	Image key	Description
—	cmdchr150	ndash	en dash
—	cmdchr151	mdash	em dash
TM	cmdchr153	trade	trademark sign
Š	cmdchr154	scaron	s caron
›	cmdchr155	rsaquo	right single angle quote
œ	cmdchr156	oelig	oe ligature
Ž	cmdchr158	zcaron	z caron
Ÿ	cmdchr159	yyuml	Y umlaut

Custom Commands

The easiest and quickest way to add functionality to the editor is to add custom commands.

Commands are sent through the `onexeccommand` event (see “`onexeccommand`” on page 131). This event sends the command to the client application or a script defined within the `external` section of the `config.xml` file.

Custom commands must follow these guidelines.

- They follow the same definition conventions as internal commands (see “Command Guidelines” on page 172).
- Icons and menu selections are created for the command as specified.
- The definition of the command’s functionality must exist within an external client application or script.
- It is the responsibility of the script/client to respond to the defined command events.
- The editor does not react to external commands other than to raise an event.

See also “External Features” on page 193 and “Creating a Custom Command” on page 25.

Button Images

Images are available to be placed on buttons. Assign an image element to a command to specify the image appears when the command is selected.

See also

- “Changing the Image that Appears on a Toolbar Button” on page 20
- “image” on page 157

Formats Supported

VisualFormat supports the following image formats.

- Windows Bitmap
- GIF
- JPEG

Sources of Images

There are two sources of images, and two kinds of image command elements.

- **Images supplied by VisualFormat** - Specify these by entering the image command's `key` attribute.

For example: `<image key="cut" />`

In this example, "cut" is the keyword that specifies the image. For a list of standard image keywords and associated images, see “Images Supplied by VisualFormat” on page 180.

-
- **Images from another source**, such as those created by your organization - specify these by entering a URL using the image command's `src` attribute. The URL can refer to a local or remote location.

For example:

```
<image src="http://www.yourcompany.com/images/mycut.gif" />
```

If an image has a key and an `src` value, the `src` attribute overrides the key.

For more information, see "Creating Your Own Images" on page 184.

Images Supplied by VisualFormat

The table on the following page lists the image keywords and associated images supplied by VisualFormat.

NOTE Note that VisualFormat also supplies a set of special characters that can appear on toolbar buttons. See "Special Character Commands" on page 177.

Keyword	Icon
about	
absmode	
ab spos	
additem	
audio	
back	
bar	
bbtn	
bgcolor	
blank	
bold	
bookmark	
borders	
borders2	
browse	
bull	

Keyword	Icon
bullet	
camera	
center	
charsmenu	
check1	
checkbox	
choice	
clean	
close	
copy	
cut	
dagger	
ddagger	
default	
delcell	
delcol	

Keyword	Icon
delete	
delrow	
details	
dropdown	
edit hyperlink	
euro	
eyeglasses	
fgcolor	
fileup	
find	
floppy	
fno f	
fontcolor	
fontcolor2	
fontcolpal	
fontface	

Keyword	Icon
fontsize	
form	
front	
glyphs	
hellip	
help	
helpwhat	
hiddenfld	
horzrule	
hyperlink	
image	
indentlf	
indentrt	
inscell	
inscol	
insert hyperlink	

Keyword	Icon
insrow	
instable	
italic	
justify	
key	
ldquo	
ldquor	
left	
link	
lock	
lsquo	
lsquor	
ltrblock	
ltredit	
mail	

Keyword	Icon
mdash	
mergecell	
ndash	
new	
newwin	
nojustify	
numbered	
oelig	
oeoelig	
one	
open	
optionbox	
page	
pagetag	
paste	
permil	

Keyword	Icon
picture	
plain	
print	
properties	
pwdfld	
rbtn	
rdquo	
redo	
removelink	
right	
rsaquo	
rsquo	
rtlblock	
rtledit	
save	
saveall	

Keyword	Icon
sbtn	
scaron	
selectall	
selectnone	
setup	
snapgrid	
space	
spellayt	
spellcheck	
splitcell	
sscaron	
strikethrough	
subscript	
superscript	
table	
tablemenu	

Keyword	Icon
textbox	
textfld	
three	
timer	
trade	
two	
underline	
undo	
up1lvl	
world	
yyuml	
zcaron	
zordermenu	
zzcaron	

Creating Your Own Images

You can create your own custom button images for the VisualFormat toolbar.

To create a new button image or modify an existing one, you can use any commercially available paint program that can produce GIF files.

By convention, button image file names start with "btn".

See *Also*: "Changing the Image that Appears on a Toolbar Button" on page 20

Image File Extensions

Although the graphic file for a toolbar button is usually a GIF (.gif) file, it can also be a Windows bitmap (.bmp), or a JPEG (.jpg) file. Windows bitmap files are larger than GIF and, therefore, take longer to download. JPEG files are optimized for photographs and images and usually do not display a small icon clearly. As a result, the GIF file format is preferred.

Size of Button Images

Although a button image can be almost any size, the standard size provided with VisualFormat is 16 by 16 pixels. If you wish, you could create buttons of a larger uniform size, as is common with Microsoft Internet Explorer, but the VisualFormat toolbar would occupy more space on your web page.

Background Color of Button Images

Also, a button image's background color should conform to the Windows' background color for buttons and other 3D objects. Any pixel that is gray (hex value C0C0C0) will display as the Windows' button (3D Objects) color.

Button Image Specification Summary

Image Attribute	Value	Comments
File Format	GIF	JPEG (JPG) and Windows Bitmap (BMP) also supported.
Width	16 pixels	Any size is possible; this is the standard size.
Height	16 pixels	Any size is possible; this is the standard size.
Background Color	RGB: 192, 192, 192; Hex: C0C0C0	Other colors do not conform to the Windows' background color.
File name prefix	btn	The prefix is only a convention, not a requirement.

Table Commands

To allow users to create tables in the editor, you must enable the table command.

To edit table specifications, use `cmdtable`.

Below is the section on the XML configuration data that enables users to create tables. (For more information about commands, see “Commands” on page 171.)

```
<table enabled="true">
    <command name="cmdtable" style="0" visible="true">
        <image key="tablemenu" />
        <caption localeRef="btnTxtTblInsrt">Table</caption>
        <toolTipText localeRef="btnTblInsrt">Insert Table</toolTipText>
    </command>
</table>
```

Fonts and Headers

The font and header commands let you specify font sizes, font styles and heading levels.

See also “Determining which Fonts, Font Sizes, and Headings are Available” on page 33.

fonts

The section that specifies the font names and sizes that users can apply to text in the editor.

The font element provides one way to define font information. However, the preferred way is to use a selections element group to generate a list of fonts and commands to set them.

Element Hierarchy

```
<config>
  <features>
    <standard>
      <fonts>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the font listing is not used.

fontname

Specifies the name of a font. The font is added to the font list made available to the user.

The name specified in the text attribute is exactly what is placed in the font tag. Be sure to use font names that browsers can interpret.

Remarks

The fontname element is one way to define a set of font names. However, the preferred method is to use a selections element group to define available font names.

Here is an example of the preferred method of defining font names.

```
<command name="cmdfontname" style="list" visible="true">
    <toolTipText localeRef="btnfntnm">Font Name</toolTipText>
    <selections name="fontnamelist" enabled="true" sorted="true">
        <listchoice>Arial, Helvetica</listchoice>
        <listchoice>Courier</listchoice>
        <listchoice>Microsoft Sans Serif, Sans Serif</listchoice>
        <listchoice>Symbol</listchoice>
        <listchoice>Times New Roman</listchoice>
        <listchoice>Verdana</listchoice>
        <listchoice>Webdings</listchoice>
    </selections>
</command>
```

Element Hierarchy

```
<config>
    <features>
        <standard>
            <font names>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the font is not used.
#text	String	""	The text that the user will view to select this font; also, the text placed into the font tag as the font name.

fontsize

Defines a font size that users can apply to text within the editor.

Remarks

Below are the font size commands available. These commands control the setting of the font size. They do not need to be defined in the feature as commands.

They are available for list box commands and scripting.

- cmdfontsize1
- cmdfontsize2
- cmdfontsize3
- cmdfontsize4
- cmdfontsize5
- cmdfontsize6
- cmdfontsize7

The fontsize element is one way to specify font sizes. However, the preferred way is to use the selections element group to generate a list of fonts and commands to set them. Here is an example use of the preferred method.

```
<command name="cmdfontsize" style="list" visible="true">
    <image key="fontsize" />
    <toolTipText localeRef="btntfontszz">Font Size</toolTipText>
    <selections name="fontsizeclist" enabled="true" sorted="true">
        <listchoice command="cmdfontsize7">7 pt</listchoice>
        <listchoice command="cmdfontsize6">6 pt</listchoice>
        <listchoice command="cmdfontsize5">5 pt</listchoice>
        <listchoice command="cmdfontsize4">4 pt</listchoice>
        <listchoice command="cmdfontsize3">3 pt</listchoice>
        <listchoice command="cmdfontsize2">2 pt</listchoice>
        <listchoice command="cmdfontsize1">1 pt</listchoice>
    </selections>
</command>
```

Element Hierarchy

```
<config>
    <features>
        <standard>
            <font size>
```

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false, the font size is not used.
localeref	String	""	The identifier to translate the #text description.
name	String	"3"	The name of the font. It must be a value from 1 through 7. 1 is the smallest font, 7 is the largest. Fonts of any other names are not used.
#text	String	""	The text that defines the font. This text appears on the selection list but is not inserted into the font tag.

headings

The section that specifies the heading levels for paragraphs.

Element Hierarchy

```
<config>
  <features>
    <standard>
      <header level>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the heading listing is not used.

heading[x]

Defines a heading level. The [x] value in the name must be between, and including, 1 and 6. Any other values are not read in.

These are the heading names.

- heading1
- heading2
- heading3
- heading4
- heading5
- heading6

Remarks

Below are the heading commands available to list boxes and buttons. These commands control the block header type.

- cmdheading1
- cmdheading2
- cmdheading3
- cmdheading4
- cmdheading5
- cmdheading6
- cmdheadingStd (returns text to normal)

They do not need to be defined in the feature as commands. They are available for list box commands and scripting.

Although you can use the heading[x] element to define block header levels, the preferred method is to use a selections element to generate a list of options with commands.

Here is an example of the preferred way of defining available header levels.

```
<command name="cmdheaderlevel" style="listbox" visible="true">
<image key="headerlevel" />
<caption localeRef="btntxthdrlvl"></caption>
<toolTipText localeRef="btndrllvl">Set the header level</toolTipText>
<selections name="headinglist" enabled="true" sorted="true">
    <listchoice command="cmdheading1" localeref="hdgtxtlv11">Heading 1</listchoice>
    <listchoice command="cmdheading2" localeref="hdgtxtlv12">Heading 2</listchoice>
    <listchoice command="cmdheading3" localeref="hdgtxtlv13">Heading 3</listchoice>
    <listchoice command="cmdheading4" localeref="hdgtxtlv14">Heading 4</listchoice>
    <listchoice command="cmdheading5" localeref="hdgtxtlv15">Heading 5</listchoice>
    <listchoice command="cmdheading6" localeref="hdgtxtlv16">Heading 6</listchoice>
    <listchoice command="cmdheadingstd" localeref="hdgtxtnorm">Normal</listchoice>
</selections>
</command>
```

Element Hierarchy

```
<config>
    <features>
        <standard>
            <headings>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the heading listing is not used.
Localeref	String	""	The translation of the #text description.
Name	String	""	These commands control the header levels. Heading1 - "heading 1" Heading2 - "heading 2" Heading3 - "heading 3" Heading4 - "heading 4" Heading5 - "heading 5" Heading6 - "heading 6"
#text	String	""	Text description of the header. This value is included in the header level listing.

External Features

Use the external feature of the XML configuration data to define external client functionality. This includes applications, Java script, and Visual Basic (VB) script.

You use command elements to define commands.

Element Hierarchy

```
<config>
  <features>
    <external>
```

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false, external commands are disabled.

Adding External Features

You can quickly add features to VisualFormat by adding functionality using Java Script or VB. External commands defined are sent up as external events. This is a powerful way to define features without requiring the development of binary modules. (See “Custom Commands” on page 178.)

Follow these guidelines when creating external features.

- Define the new functionality as a command within the external section of the XML configuration data.
- This section acts as the definition for the External Event feature.
- Follow all rules for defining standard features.

Examples

Here is where you would define the external feature within the XML configuration data.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro" version="2" revision="1">
<features>
    <external enabled="true">
        <!--This is where script/client commands are defined that
            will go back up to the client as events. -->
        . .
    </external>
    . .
</features>
```

Here is an example of the external script/client command definition.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro" version="2" revision="1">
    <features>
        <external enabled="true">
            <!--This is a user defined command that will go back up to the script-->
            <command name="saveaspdf" style="0" visible="true">
                <image key="http://site.com/images/pdf.gif"/>
                <caption localeRef="btnScrPdf"> Save as a PDF file.</caption>
                <toolTipText localeRef="btnPdf">Saves as PDF.</toolTipText>
            </command>
            .
        </external>
    </features>
    .
</config>
```

Here is an example of a custom module creating its own section within the features.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro" version="2" revision="1">
<features>
    <pdfgenerator enabled="true">
        <!--This is a user defined command sent to a DLL-->
        <command name="saveaspdf" style="0" visible="true">
            <image key="http://site.com/images/pdf.gif"/>
            <caption localeRef="btnScrPdf">Safe as a PDF file.</caption>
            <toolTipText localeRef="btnPdf">Converts to PDF.</toolTipText>
        </command>
    </pdfgenerator>
</features>
```

Viewing and Editing HTML Content

This section describes elements that let users view and edit the HTML content of their web page.

The ViewAs Feature

The ViewAs feature determines whether or not users can view the HTML source code. If you allow users to view source code, they do so by right-clicking on the mouse while the cursor is in the editor. When they do, two menu options appear.

- **View HTML** - lets the user view the source code for the web page
- **View WYSIWYG** - returns the user to edit mode

If you allow users to view source code, you can further specify whether users can view only the body of the page or the entire page including the header.

Element Hierarchy

```
<config>
  <features>
    <viewas>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, users cannot view the content in different formats.
mode	String	Body	Specifies how much of the source code appears when the user views HTML. This attribute has two values. " Body " - only the body of the document " Whole " - the entire source, including headers.
publish	String	Cleanhtml	The level of cleanliness applied when the user chooses ViewHTML. The higher the level, the potentially more time to process the source. This attribute has three values. " Minimal " - General tag organization " Cleanhtml " - Removes overlapping tags and merges font tags. " Xhtml " - HTML level of organization

The EditHTML Feature

The EditHTML feature determines whether or not users can edit the HTML source of the content.

Element Hierarchy

```
<config>
  <features>
    <edithtml>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the editHTML feature is disabled.

Cleaning HTML

VisualFormat provides the following elements that prepare the HTML code of the web content for publishing.

- clean
- remove, a sub element of clean
- endtag, attribute, tagonly and tagelement: sub-elements of remove

The clean feature defines general HTML clean-up features, such as the quality of the HTML code output by the editor.

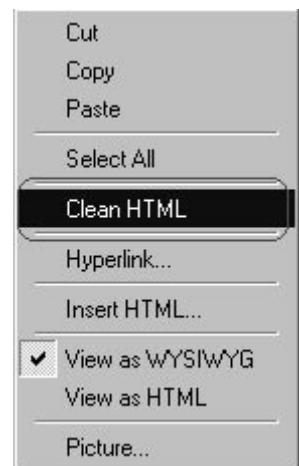
You can use Remove, Endtag, Attribute, Tagonly and Tagelement elements of the clean feature to remove specific elements from the content when it is cleaned.

Providing User Access to the Clean Feature

By default, the command that launches the clean feature (`cmdclean`) appears on the context-sensitive menu.

However, you can assign the command to a button, just as you can assign any other command to a button (see “Commands” on page 171).

The user also receives an option to clean HTML when pasting content from Microsoft Word 2000.



Clean Feature

The clean HTML feature ensures that the content loaded into the editor is readable and concise HTML source code.

Element Hierarchy

```
<config>
  <features>
    <clean>
```

Attributes

Name	Attribute Type	Default	Description
charencode	String	"charref"	<p>Determines how to encode special and extended characters. Five values are available.</p> <ul style="list-style-type: none">• binary• entityname• charref• special• latin <p>For more information, see "Encoding Special Characters" on page 207.</p> <p>See Also: "preservechars" on page 199</p>

Name	Attribute Type	Default	Description
cr	String	"cr"	<p>How to translate the carriage return character when the content is saved. Four values are available.</p> <p>"" - remove cr character</p> <p>"cr" - do not process cr characters</p> <p>"charref" - replace cr with its character reference, that is &#13;</p> <p>"\r" - replace cr with \r</p>
If	String	"If"	<p>How to translate the line feed character when the content is saved. Four values are available.</p> <p>"" - remove If character</p> <p>"If" - do not process a If character</p> <p>"charref" - replace If character with its character reference, that is &#10;</p> <p>"\n" - replace If character with \n</p>
preferfont	Boolean	"false"	<p>If "true," span tags with font styles are converted to font tags.</p> <p>If a font name, color, or size are specified using a span tag (for example, in content pasted from MS Word), the span tag can be converted to a font tag. Font tags are compatible with older browsers and allow font attributes to be easily edited in eWebEditPro.</p>
reducetags	Boolean	"false"	<p>Whether eWebEditPro eliminates unnecessary tags.</p> <p>When a user pastes content from other applications into eWebEditPro, the content may contain redundant tags, such as extra font and bold tags. The extra tags can be combined or safely removed.</p>
preservechars	String	""	<p>Identifies characters that will not be converted to character references. Conversion of these characters is done on the server side only.</p> <p>When entering values for this element, enter character references if XML requires them. Refer to an XML reference to determine which characters require conversion, and how to convert them.</p> <p>For example, to prevent the "less than" (<) and "greater than" (>) characters from being converted to their character references, enter</p> <pre>preservechars value="&lt;&gt;"</pre> <p>For more information about special characters, see "Encoding Special Characters" on page 207.</p>

Name	Attribute Type	Default	Description
showonsize	integer		<p>The minimum number of characters of HTML code needed to display a dialog box that appears when the user saves content. The dialog indicates that eWebEditPro is examining and cleaning HTML.</p> <p>This attribute prevents the dialog box from flashing when there is little or no content.</p> <hr/> <p>Note: This attribute does not appear in the configuration data by default. You must enter the attribute name and value to use it.</p> <hr/>

Remove Element

This rule defines what elements are removed from the content when it is cleaned.

Element Hierarchy

```
<config>
  <features>
    <clean>
      <remove>
```

Attributes

Name	Attribute Type	Default	Description
#text	String		There are no attributes to the remove element.

Remove Endtag Element

This rule defines what elements are globally removed from the content when it is cleaned. Some HTML end tags are optional and can be removed with little risk.

In general, this option is not recommended, but there may be situations in which certain end tags (e.g., </p>) are not desired.

VisualFormat removes end tags when the content is saved.

This option is ignored if the `publish` attribute of the `clean` command is set to **xhtml**.

You can only enter one `<remove>` element, but you can enter several `<endtag>` elements.

Element Hierarchy

```
<config>
  <features>
    <clean>
      <remove>
        <endtag>
```

Attribute

Name	Attribute Type	Default	Description
#text	String		The tag to remove.

Example

```
<clean cr="cr" lf="lf" autodetect="yes" publish="minimal">
  <remove>
    <endtag>p</endtag>
    <endtag>li</endtag>
  </remove>
</clean>
```

Remove Attribute Element Feature

When the user cleans the content, that procedure can remove attributes globally from the content.

In general, this option is not recommended, but there may be situations in which you want to remove certain attributes (for example, `id`, `onclick`, etc.).

VisualFormat removes attributes when the content is saved.

This option is ignored if the `publish` attribute of the `clean` command is set to **xhtml**.

You can only enter one <remove> element, but you can enter several <attribute> elements within it.

Element Hierarchy

```
<config>
  <features>
    <clean>
      <remove>
        <attribute>
```

Attribute

Name	Attribute Type	Default	Description
#text	String		The attribute to remove.

Example

```
<clean cr="cr" lf="lf" autodetect="yes" publish="minimal">
  <remove>
    <attribute>onclick</attribute>
  </remove>
</clean>
```

Removing Tags and Content Between Them

When the user cleans the content, that procedure can remove specified HTML tags only, or specified tags along with any content between them.

For example, you can set up the clean command to remove all font tags, image (img) tags, and script elements.

You can only enter one <remove> element, but you can enter several <tagonly> and <tagelement> elements within it.

Element Hierarchy

```
<config>
  <features>
    <clean>
      <remove>
        <tagonly>
        <tagelement>
```

Attribute

Name	Attribute Type	Default	Description
tagonly	String		The tag to remove. Only the tags are removed.
tagelement	String		The tag to remove. All content between the tags is also removed.

Example

```
<clean ...>
    <remove>
        <tagonly>font</tagonly>
        <tagelement>script</tagelement>
    </remove>
</clean>
```

The Spellcheck Feature

The spellcheck feature controls the operation of spell checking within the client. The feature has three commands, summarized below.

Command	Lets You Specify
Spellcheck	Enables the spell check as-you-type feature.
Spellayt	When spell check as-you-type begins; location of image file that marks misspelled words.
Spellingsuggestion	Number of correctly spelled words similar to a misspelled word that appear.

Spellcheck

Defines whether or not the spell check feature operates.

Element Hierarchy

```
<config>
  <features>
    <spellcheck>
```

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false, spell checking is disabled.

Spellayt

Defines how spell checking as-you-type operates.

Element Hierarchy

```
<config>
  <features>
    <spellcheck>
      <spellayt>
```

Attributes

Name	Values	Default	Description
autostart	Boolean	True	If true, spell check starts to check spelling "as-you-type" as soon as possible without user intervention. The editor will take longer to launch due to spell checking. If false, the user must press the button or menu option to activate spell check as-you-type.
enabled	Boolean	True	If false, auto-spell checking is disabled.
markmisspelledsrc	String	""	Specifies the URL of the graphic file (by default, a wavy red line) that marks misspelled words. The name of the file provided is wavyred.gif. The default value ("") resolves itself to the location of the XML configuration data. This is interpreted as a path, so the case is maintained.

Spellingsuggestion

Defines how suggestions for correcting spelling errors operates.

Element Hierarchy

```
<config>
  <features>
    <spellcheck>
      <spellingsuggestion>
```

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false, the spell checker does not suggest words to replace a misspelled word.
max	Integer	20	The maximum number of correctly-spelled words that appear after the spell checker finds a misspelled word. To view this list, right click the mouse.

Example of Spell Check Features

The following is the default version of the spell check features in the XML configuration data.

```
<spellcheck enabled="true">
    <spellayt autostart="false" enabled="true"
        markmisspelledsrc="[eWebEditProPath]/wavyred.gif"/>
    <spellingsuggestion enabled="true" max="4"/>
    <command name="cmdspellayt" style="toggle" enabled="true">
        <image key="spellayt"/>
        <caption localeRef="btnTxtSplayt">Check Spelling As You Type</caption>
        <toolTipText localeRef="btnSplayt">Check Spelling As You Type </toolTipText>
    </command>
    <command name="cmdspellcheck" enabled="true">
        <image key="spellcheck"/>
        <caption localeRef="btnTxtSplck">Check Spelling</caption>
        <toolTipText localeRef="btnSplck">Check Spelling</toolTipText>
    </command>
</spellcheck>
```

Encoding Special Characters

Factors that Affect the Display of Special Characters

The HTML specification defines special characters for a set of punctuation symbols, accented letters, and a variety of non-Latin characters. As the HTML specification has changed, so has browser support for special characters.

For example, Microsoft defined several special characters that previously displayed only in Internet Explorer on Windows. They are extended characters that map to binary values 128 to 159. Depending on the browser version and operating system, the characters may appear as expected, as a question mark (?), or as a small rectangle. The W3C has adopted most of these extended characters in HTML 4, but mapped them to different binary values.

Using the wrong font can also prevent the proper display of a character. This is a common problem when copying from Microsoft Word, where many special characters are in the Symbol font. If this font is not available in the browser or not permitted in the editor, special characters do not display properly.

For example, the Euro symbol was designed for the European Economic Community (EEC) in the late 1990s. Obviously, operating systems and browsers created earlier could not display it.

Euro character (shown using an image)	€
Euro in Verdana font (display depends on your browser)	€
Euro in Courier New font (display depends on your browser)	€
Entity Name	€
Microsoft Windows Extended Character Reference	€
HTML 4 Character Reference	€

Characters with binary values 160 to 255 are also special characters because they display differently depending on the browser's language (or locale) and the `charset` attribute in the meta tag on the web page. Below is an example meta tag.

```
<meta http-equiv=Content-Type content="text/html; charset=iso-8859-2">
```

The display of special characters can also be controlled from the browser. For example

- in IE 5, from the menu bar, select **View > Encoding >** language of your choice. (You may need to install the IE option for international language support).
- in Netscape 4.7, select **View > Character Set >** language of your choice

In each case, the possible languages are grouped as West European (Latin1), East European (Latin2), Cyrillic, Arabic, Greek, Hebrew, and more. Each character set is defined by ISO 8859, a standard for coded graphic character sets established by the International Organization for Standardization.

The ISO 8859 special characters are listed below. When viewed in a browser, these characters display differently if you change your browser's encoding.

¡ ¢ £ ¤ ¥ ¡ § ¨ © ª « ¬ - ® ¯

° ± ² ³ ´ μ ¶ · , ¹ º » ¼ ½ ¾ ¸

À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï

Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý þ ß

à á â ã ä å æ ç è é ê ë ì í î ï

ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

In summary, the following factors affect the display of special characters.

- browser and browser version
- operating system

-
- language of the operating system (English, Polish, Arabic, etc.)
 - font (Times, Arial, Helvetica, Symbol, etc.)
 - charset attribute in the meta tag (windows-1252, iso-8859-1, etc.)
 - encoding/character set setting of the browser (Western, Central European, UTF-8, etc.)

Displaying Asian Languages

Many Asian languages, such as Japanese, Korean, and Chinese, are represented by two bytes instead of one. The binary values for these characters are in the range 256 to 65535. These are mapped as Unicode characters.

VisualFormat can optionally convert these characters to their character reference or leave them as double-byte binary Unicode values (not UTF-8). For example, a character whose binary value is 1234 converts to Ӓ.

Unicode Characters

Unicode characters (double byte characters typically used for Asian languages) are normally converted to character references, for example, Ӓ. To output Unicode characters as their double-byte binary value, set the charencode attribute to **binary**. If your site uses UTF-8 encoding, you can set the charencode attribute to **utf-8** instead of binary, but the two are essentially the same.

Configuring VisualFormat for Extended and Special Characters

VisualFormat can be configured to represent extended and special characters in several ways. They are

- **binary** - extended, special, and double-byte characters as binary (Unicode, which can be converted to UTF-8)
- **entityname** - extended and special characters as their entity name; double-byte characters as their character reference

-
- **charref** - extended, special, and double-byte characters as their character reference. This is the default value.
 - **special** - extended characters as their entity name; special characters as binary; double-byte characters as their character reference
 - **latin** - extended characters as HTML 4 character references; special characters as binary; double-byte characters as their character reference

charencode Attribute

To configure VisualFormat, set the `charencode` attribute of the `clean` tag in the XML configuration data. For example,

```
<!-- values for charencode: utf-8, binary, entityname, charref, special, latin -->
<clean charencode="charref" ...>
```

NOTE To prevent selected characters from being converted to character references, use the `preservechars` attribute of the `clean` command. For more information, see “`preservechars`” on page 199.

The values for charencode and their effect are described in the following table.

Value of charencode	Description	Sample
entityname	<p>Extended characters are represented using their entity name (e.g., &euro;) where possible.</p> <p>Special characters are represented using their entity name (e.g., &nbs; or &Agrave;).</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <p>80: &euro; &sbquo; &fnof; &bdquo; &hellip; &dagger; &Dagger; &circ; &permil; &Scaron; &lsaquo; &OElig; &#381;</p> <p>90: &lsquo; &rsquo; &ldquo; &rdquo; &bull; &ndash; &mdash; &tilde; &trade; &scaron; &rsaquo; &oelig; &#382; &Yuml;</p> <p>Special Characters: (Latin1 shown)</p> <p>A0: &nbs; &excl; &cent; &pound; &curren; &yen; &brvbar; &sect; &uml; &copy; &ordf; &laquo; &not; &shy; &reg; &macr;</p> <p>B0: &deg; &plusmn; &sup2; &sup3; &acute; &micro; &para; &middot; &cedil; &sup1; &ordm; &raquo; &frac14; &frac12; &frac34; &iquest;</p> <p>C0: &Agrave; &iacute; &Acirc; &tilde; &Auml; &Aring; &AElig; &Ccedil; &Egrave; &Eacute; &Ecirc; &Euml; &Igrave; &Iacute; &Icirc; &Iuml;</p>
charref	<p>Extended characters are represented using their HTML 4 character reference (e.g., &#8364;).</p> <p>Special characters are represented using their character reference (e.g., &#160; or &#192;).</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <p>80: &#8364; &#8218; &#402; &#8222; &#8230; &#8224; &#8225; &#710; &#8240; &#352; &#8249; &#338; &#381;</p> <p>90: &#8216; &#8217; &#8220; &#8221; &#8226; &#8211; &#8212; &#732; &#8482; &#353; &#8250; &#339; &#382; &#376;</p> <p>Special Characters: (Latin1 shown)</p> <p>A0: &#160; &#161; &#162; &#163; &#164; &#165; &#166; &#167; &#168; &#169; &#170; &#171; &#172; &#173; &#174; &#175;</p> <p>B0: &#176; &#177; &#178; &#179; &#180; &#181; &#182; &#183; &#184; &#185; &#186; &#187; &#188; &#189; &#190; &#191;</p> <p>C0: &#192; &#193; &#194; &#195; &#196; &#197; &#198; &#199; &#200; &#201; &#202; &#203; &#204; &#205; &#206; &#207;</p> <p>D0: &#208; &#209; &#210; &#211; &#212; &#213; &#214; &#215; &#216; &#217; &#218; &#219; &#220; &#221; &#222; &#223;</p>

Value of charencode	Description	Sample
special	<p>Extended characters are represented using their entity name (e.g., &euro;) where possible.</p> <p>Special characters remain as binary, except the non-breaking space, which is represented as &nbsp;.</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <pre> 80: &euro; &sbquo; &fnof; &bdquo; &hellip; 1dagger; &Dagger; &circ; &permil; &Scaron; 1saquo; &OElig; &#381; 90: &lsquo; &rsquo; &ldquo; &rdquo; &bull; 1ndash; &mdash; &tilde; &trade; &scaron; 1rsaquo; &oelig; &#382; &Yuml;</pre> <p>Special Characters: (Latin1 shown)</p> <pre> A0: &nbsp; ; ¢ £ ¥ ; \$ “ ” ® “ ” – ® B0: ° ± ± ‘ μ ¶ . , , ° » ¶ » % ¢ C0: à Á Â Ã Ä Å Ç È É Ê Ë Ì Í Í D0: Đ Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß E0: à á â ã ä å ç è é ê ë ì í ï F0: ö ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ</pre>
latin	<p>Extended characters are represented using their HTML 4 character reference (e.g., &#8364;).</p> <p>Special characters remain as binary, except the non-breaking space, which is represented as &#160;.</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <pre> 80: &#8364; &#8218; &#402; &#8222; &#8230; &#8224; &#8225; &#710; &#8240; &#352; &#8249; &#338; &#381; 90: &#8216; &#8217; &#8220; &#8221; &#8226; &#8211; &#8212; &#732; &#8482; &#353; &#8250; &#339; &#382; &#376;</pre> <p>Special Characters: (Latin1 shown)</p> <pre> A0: &#160; ; ¢ £ ¥ ; \$ “ ” ® “ ” – ® B0: ° ± ± ‘ μ ¶ . , , ° » ¶ » % ¢ C0: à Á Â Ã Ä Å Ç È É Ê Ë Ì Í Í D0: Đ Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß E0: à á â ã ä å ç è é ê ë ì í ï F0: ö ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ</pre>

Choosing a charencode Value

The best charencode value to use depends on the environment in which the content is viewed and personal preference for entity names versus character references. Here are some examples.

- If the environment only supports 7-bit ASCII characters (for example, a database), you must use entityname or charref.

- Values of special or latin create smaller file sizes, because the special characters require one byte instead of six or more bytes to represent each character.
- A value of binary create the smallest file sizes for content that consists mostly of Asian characters (e.g., Japanese, Korean, Chinese), because the characters require just two bytes instead of seven or more.
- Some sites convert Unicode characters to a byte stream format of UTF-8. If your site consistently uses UTF-8, use a value of utf-8.

The following table recommends `charencode` values for certain conditions.

Condition	Recommended charencode Value	Comments
Database supports only 7 bit characters	entityname or charref	Extended and special characters will be corrupted if wrong charencode value is selected. Your choice depends on your preference for entity names or character references.
Database supports only 8 bit characters	any except binary; use utf-8 only if your site uses UTF-8 consistently	Some special and all double-byte characters are corrupted if you choose binary.
Double-byte encoding, typically for an Asian language, and document size is important	binary or utf-8	Database must support Unicode (double-byte) characters. Note: Unicode is not the same as UTF-8.
Entity names are preferred	entityname	Extended and special characters are their entity name.
Entity names are preferred, but in a non-Western European language	special	Special characters are binary for different document encodings, but extended characters are their entity name.
ISO-8859 (Latin) or windows charset encoding on document, but not Latin1 (that is, not windows-1252 or iso-8859-1)	latin or special	Your choice depends on your preference for entity names or character references for extended characters.

Condition	Recommended charencode Value	Comments
Netscape Navigator 4 used for browsing	charref	Most extended and special characters appear. Double-byte characters do not appear if the browser or operating system does not support the language. If another charencode value is selected, some extended and special characters may appear as a question mark (?) or their entity name.
UTF-8 charset encoding on document	entityname or charref; use utf-8 only if your site uses UTF-8 consistently	Special and double-byte characters do not display correctly as binary. Your choice depends on your preference for entity names or character references.
XML without XHTML DTD/Schema	charref; use utf-8 only if your site uses UTF-8 consistently	XML supports only a very limited set of entity names unless the XHTML (or other) DTD is provided.
Not sure	charref	charref works with both UTF-8 encoding and XML parsers. It also gives the best results in Netscape. If special characters always appear as West European letters instead of the proper language, try latin.

References

Character entity references in HTML 4 (<http://www.w3.org/TR/REC-html40/sgml/entities.html>)

The ISO 8859 Alphabet Soup (<http://czyborra.com/charsets/iso8859.html>)

Dan's Web Tips: Characters and Fonts (<http://www.dantobias.com/webtips/char.html>)

Implementing a Web Site that Uses UTF-8 Encoding

Background Information

UTF-8 is a byte stream encoding scheme that converts each double-byte Unicode character to one, two or three bytes.

See Also: "Encoding Special Characters" on page 207.

For example, the letter a has an ASCII value of 97 (61 hex). It maps unchanged in UTF-8 to a single byte with a value of 97.

The single quote character (‘) has an ASCII value of 231 (E7 hex). It converts to two UTF-8 bytes: 195 167 (C3 A7 hex).

The Japanese single quote character (‘) has a Unicode value of 27231 (6A5F hex). It converts to three UTF-8 bytes: 230 169 159 (E6 A9 9F hex).

Implementing UTF-8

To implement UTF-8, follow these points.

- All web pages that include the editor or that display the content must set the charset to UTF-8.

```
<head>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
...
</head>
```

- If you are using a database, ensure that it can accept UTF-8 or Unicode characters.
- Set the XML configuration data to produce characters for UTF-8.

```
<clean charencode="utf-8" ...>
```

(For more information, see "charencode Attribute" on page 210.)

- Load UTF-8 encoded content into the hidden field for the editor. How you load this content varies according to your server platform and environment.

```
<input type=hidden name="MyContent1" value="Content that is UTF-8 and HTML encoded">
```

NOTE

You may not be able to use standard HTML encoding functions, such as `HTMLEncode()` in ASP.

WARNING!

Content stored in JavaScript string variables and in the VisualFormat ActiveX control is stored as Unicode (double-byte) characters. When a browser reads the value of the hidden field, the browser converts the UTF-8 byte stream to a Unicode string for JavaScript. Similarly, when a form is posted to the server, the browser converts content stored in the hidden field to UTF-8.

TIPS

- If the UTF-8 byte stream is treated as a Unicode string, the special characters are corrupted and appear as two or three characters.
- If a Unicode string is interpreted as UTF-8, special characters are corrupted, and the number of characters is reduced, thereby eliminating some characters whether or not they are special.
- ASCII characters (A-Z, a-z, 0-9, etc.) always appear correctly because they have the same value in Unicode and UTF-8.

Setting the charset Parameter

If you are retrieving the entire document from the editor, set the charset parameter to **utf-8**. If you are retrieving only the body contents, you may still set the charset parameter.

To set the charset parameter to **utf-8**, update the `ewebeditprodefaults.js` file so that charset is set to **utf-8**. Or, you can use JavaScript to modify `eWebEditPro.parameters` on the page using this code.

```
eWebEditPro.parameters.editor.charset = "utf-8";
```

Browser Support for UTF-8

In order for the browser to support UTF-8, the following conditions must exist.

- The browser displaying the editor must support UTF-8.
 - Microsoft provides language add-ons for Internet Explorer.
 - Because Netscape 4.7x may not display Asian characters on English Windows (they may appear as question marks '?'), you need a language-specific version of the operating system and/or browser.
 - Netscape 6 supports multiple languages.
- Ensure that the browser encoding uses UTF-8. Set the browser to unicode using the sequence of menu options indicated below.
 - Internet Explorer:
View > Encoding > Auto-Select or Unicode (UTF-8)
 - Netscape 4.7:
View > Character Set > Unicode (UTF-8)
 - Netscape 6:
View > Character Coding > Auto-Detect > Auto-Detect (All) or Unicode (UTF-8)

For More Information about UTF-8

- UTF-8 (technical specification) - <http://www.ietf.org/rfc/rfc2279.txt>
- The ISO 8859 Alphabet Soup - <http://czyborra.com/charsets/iso8859.html>
- Dan's Web Tips: Characters and Fonts - <http://www.dantobias.com/webtips/char.html>

Style Sheets

A style sheet is a file (extension .css) that contains specifications for the visual elements of a web page, such as heading sizes, fonts and margins. You use a style sheet to override default HTML values for these elements on a group of web documents or an entire web site.

Style sheets let you establish a set of style specifications and apply them to all pages. Assume, for example, that the default display for the <H3> tag is Times New Roman.

Heading 3 default

If you apply a style sheet, it might modify the <H3> tag, like this.

```
h3 {FONT-FAMILY: Arial; FONT-SIZE: 14pt; MARGIN: 12pt 0in 3pt}
```

The text follows the style sheet specifications, and looks like this.

Heading 3 default

As a result, a web site containing thousands of pages and updated by scores of editors can have a consistent look.

A good web site that explains style sheets is <http://www.w3schools.com/css/default.asp>.

This section explains the following topics relating to using style sheets with VisualFormat.

- Using Style Sheets to Standardize Formatting
- The Default Style Sheet
- Applying Style Sheets
- The BodyStyle Parameter
- Preserving Tags When Office Content is Pasted

-
- Saving Style Sheet Tags When Content is Saved
 - Implementing Style Class Selectors

Using Style Sheets to Standardize Formatting

You can combine a style sheet with the toolbar configuration procedures (see “Defining the Toolbar” on page 12) to control the formatting of the content that users produce.

For example, you could remove from the toolbar the menu options that let users select font size, color and style. Then, in a style sheet, you would specify a font size, color and style. If you make these modifications, users can enter text but not change its size, color or style -- the style sheet has standardized those specifications.

NOTE The bodyStyle parameter also lets you apply style sheet attributes to the content. See “bodyStyle” on page 117.

The Default Style Sheet

VisualFormat provides a default style sheet, ektnormal.css, that emulates the Word 2000 Normal.dot template. If you assign this style sheet in the XML configuration data, the Word 2000 default styles are applied to the content.

To do this, set the href attribute in the features > standard > style section of the XML configuration data to look like this.

```
<features>
  <standard autoclean="true" publish="xhtml">
    <style publishstyles="true" href="[eWebEditProPath]/ektnormal.css"/>
```

Applying Style Sheets

You can create your own style sheet and apply it to the VisualFormat editor. There are three levels at which you can apply a style sheet.

-
- **the XML configuration data** - affects all editors that refer to it (see “The XML Configuration Data” on page 132).

NOTE If your eWebEditPropages refer to several config.xml files (for example, you have different files for different user groups), and you want all pages to use the same style sheet, assign the same style sheet in all of the XML configuration data.

- **a page** - affects only the editors on one page
- **a single occurrence of the editor** - affects only one instance of the editor

WARNING! Depending on your settings, you probably also need to specify the style sheet when the content is published. If you use templates in your web application (for example, a content management system), a reference to the style sheet is required. This is typically done using the link tag. For example

```
<link rel="stylesheet" type="text/css" href="/ewebeditpro2/xyz.css">
```

Note that if a style sheet is specified in more than one location, the most local one takes precedence. For example, if a sheet is specified in all three locations listed above, the style sheet applied to the single occurrence of the editor would be used.

If the most local style sheet does not includes a specification for a certain tag, the browser will display its default for that tag -- it does not look in higher level style sheets for that tag's specifications.

Specifying a Style Sheet in the XML Configuration Data

You assign a style sheet using the `style` tag of the XML configuration data. To implement a style sheet in the XML configuration data, follow these steps.

You can also apply, list and disable style sheets using ActiveX methods. For more information, see “ActiveX Style Sheet Methods” on page 124.

1. Create your style sheet file (for example, xyz.css).
2. Open the config.xml file in the directory where you installed VisualFormat.
3. Move to the `style` tag, located within the `features > standard` section of the XML configuration data.

```
<features>
```

```
<standard autoclean="true" publish="xhtml">
<stylepublishstyles="true" href="[eWebEditProPath]/ektnormal.css"/>
```

Note that [eWebEditProPath] refers to the eWebEditProPath variable in the ewebeditpro.js file. If your style sheet resides in a different directory, replace [eWebEditProPath] with the directory pathway.

4. Change the href attribute in the style command so that it refers to your style sheet.

```
<style publishstyles="false" href="/yourpath/xyz.css"/>
```

5. Set publishstyles to **false**.

```
<style publishstyles="false" href="/yourpath/xyz.css"/>
```

Adding a Style Sheet to a Single Page

1. Open the page to which you want to add a style sheet.
2. Set the styleSheet parameter by adding JavaScript to the page before the editor is created.

```
<script language="JavaScript1.2">
<!--
eWebEditPro.parameters.styleSheet = "/yourpath/xyz.css";
// --
</script>
<!-- code to place the editor on the page goes here -->
```

Dynamically Changing a Style Sheet for a Single Instance of the Editor

1. Open the page to which you want to add a style sheet.
2. Add the following JavaScript function below the page's head tag.

```
<script language="JavaScript">
function setStyleSheet(strEditorName, strCSS)
{
    eWebEditPro[strEditorName].setProperty("StyleSheet", strCSS);
}
</script>
```

Replace strEditorName with the name of the editor, and strCSS with the name of the style sheet.

3. On the page where you create VisualFormat, set the onready event to call the setStyleSheet function.

For example,

```
<script language="JavaScript">
eWebEditPro.onready = "setStyleSheet(eWebEditPro.event.srcName, '/yourpath/xyz.css')";
</script>
```

Tip: You can set the `StyleSheet` property to change the style sheet after the editor loads. For example, you might want to change the style sheet when the user picks from a list of styles that you provide.

The BodyStyle Parameter

The `BodyStyle` parameter also affects all editors, or an instance of the editor. If the `body` style parameter is set, it takes precedence over a style sheet. The parameter applies the style to the `style` attribute of the body tag.

For more information, see “`bodyStyle`” on page 117.

Preserving Tags When Office Content is Pasted

Within the XML configuration data, the `style` tag has a `preservewordstyles` attribute that determines whether class and style HTML tags are preserved when Microsoft Office 2000 content is pasted into the editor.

If you set this attribute to “**true**”, class and style attributes are preserved when pasting Word 2000 content. If set to “**false**”, the class and style attributes are removed.

Below is an example of how to implement this feature within the configuration data.

```
<features>
  </external>
  <standard autoclean="true" publish="xhtml">
    <style preservewordstyles="true"/>
```

Saving Style Sheet Tags When Content is Saved

Within the XML configuration data, the `style` tag has a `publishstyles` attribute that determines whether the style sheet specifications for each tag are inserted into the file *when the content is saved*.

```
<features>
</external>
<standard autoclean="true" publish="xhtml">
<style publishstyles="true"/>
```

Below is an example of the html text of a saved line when publishstyles is set to **true**.

```
<p align="center" style="BOTTOM: 0px; FILTER:; FONT-FAMILY: 'Times New Roman'; FONT-SIZE: 12pt; MARGIN: 0in 0pt"> VARs benefits and features</p>
```

Here is the same line when publishstyles is set to **false**.

```
<p> VARs benefits and features </p>
```

Setting Publishstyles to True

Set publishstyles to **true** to make sure that the formatting specifications remain with the content after it is saved.

Setting Publishstyles to False

Set publishstyles to **false** to maintain control of the styles for an entire web site. In this case, you would not want to insert style sheet specifications for each tag. Instead, your style specifications would be taken from the style sheet specified in the display page's head tags or, if you are using a content management system, from the template file.

Another advantage of setting publishstyles to **false** is that it greatly reduces the size of the html page (as you can see from the example above).

Implementing Style Class Selectors

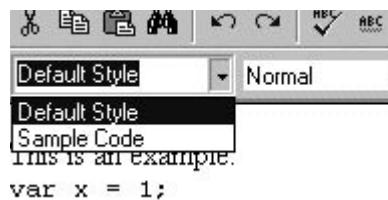
A drop-down toolbar menu (`cmdselstyle`) lets users choose a style class from a list and apply the class to selected text.

Only styles whose `visible` attribute is set to “**true**” (the default value) appear on the list. The styles appear in the order in which they are entered into the style sheet assigned to the editor.

As an example of using style class selectors, assume that your web site features text that is sample programming code. The

Webmaster would open the organization's style sheet, create a style class called "Sample Code" and assign appropriate formatting specifications to it (such as `font-size:9.0pt; font-family:"Courier New"`).

Then, when a user types sample programming code as text into the editor, he could select the text, click the drop-down menu and select **Sample Code** from the list (see illustration).



The HTML code for this line would look like this:

```
<p class="code">var 1 = x;</p>
```

The rest of this section explains

- types of style classes
- translating styles to a foreign language
- suppressing styles from the drop-down list

Types of Style Classes

There are two types of style classes.

Type	Example	Can be applied to
with tag specifiers	<code>p.box { border: solid 2px red }</code>	Only HTML tags specified in the definition. The style in this example can only be applied to text surrounded by <code><p></code> tags. Affects entire paragraph.
without tag specifiers	<code>.highlight { background-color: yellow; }</code>	Selected text using the <code></code> tag, regardless of tags surrounding the text. Affects selected text only.

The following sample code illustrates both kinds of style classes.

```
<p class="box">IMPORTANT: read this <span class="highlight">highlighted</span> word</p>
```

Translating Style Classes to a Foreign Language

You can translate the drop-down menu list so that non-English speaking users see it in their native language. To accomplish this, assign a localeRef attribute and code to a style class. For example

```
.code {  
localeRef:cssCode;
```

Then, translate the code to a foreign term in the appropriate locale.xml file. When the editor displays the list, it displays the style names from the locale file. (For more information, see “Translating Button Captions and Tool Tips to a Foreign Language” on page 23.)

For example, assume that your users speak French, so you would modify the locale040c.xml localization file. Also, assume that the style “Sample Code” translates into “Code d'échantillon” in French.

Here is an example of a *standard* style sheet specification. (The red is added for emphasis.)

```
.code {  
caption:Sample Code;  
margin:0in;  
font-size:10.0pt;  
font-family:"Courier New";}
```

Here is a style sheet specification with a reference to a localeref.

```
.code {  
localeRef:cssCode;  
margin:0in;  
font-size:10.0pt;  
font-family:"Courier New";}
```

Here is how to update the locale040c.xml localization file so that it displays “Code d'échantillon” on the drop-down list.

```
<cssCode>Code d'échantillon</cssCode>
```

Suppressing Styles from the Drop-down List

If you want to suppress styles from the drop-down list, add `visible:false` to the style class's definition in the style sheet. For example

```
.code {  
visible:false;  
margin:0in;  
font-size:10.0pt;  
font-family:"Courier New";}
```

Managing Hyperlinks

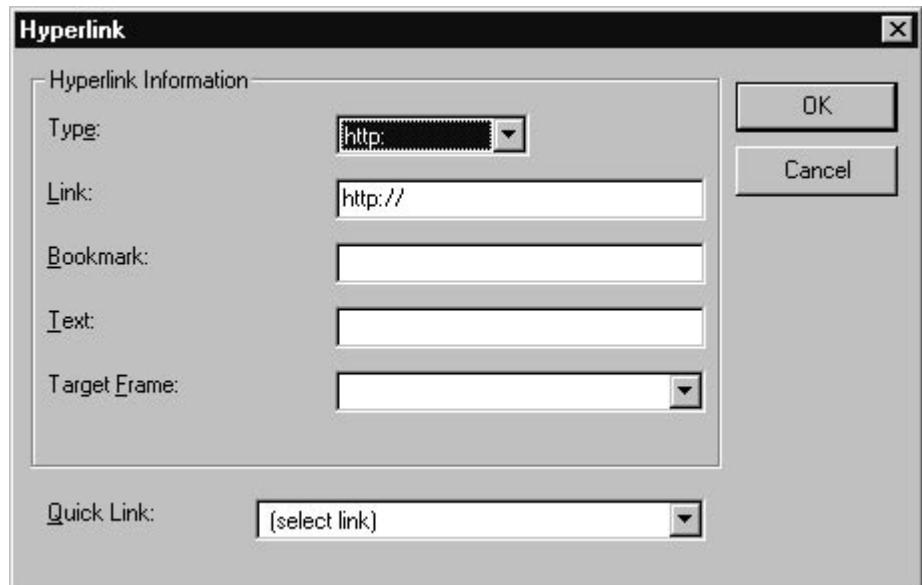
VisualFormat's standard toolbar features three buttons that let users manage hyperlinks within their content.

- The *New Hyperlink* toolbar button () lets users quickly add a hyperlink to their web content
- The *Edit Hyperlink* toolbar button () lets users change information about a hyperlink
- The *Remove Hyperlink* toolbar button () lets users remove a hyperlink

This section explains how to customize the dropdown menus in the Edit Hyperlink dialog box.

Customizing Dropdown Menus in the Hyperlink Dialog Box

This section explains how to customize the Hyperlink dialog box (illustrated below).



Specifically, the section explains how to customize the values that appear in dropdown lists at the following fields.

- **Type**
- **Target Frame**
- **Quick Link**

You edit these lists within the XML configuration data, under command `name="cmdhyperlink"`.

Customizing each list is explained below.

Customizing the Lists of the Hyperlink Dialog Box

By default, these lists are not part of the XML configuration data. As a result, you must add each list to the XML configuration data that you want to customize.

After you add each list to the XML configuration data that you want to edit, customize the list by

- adding or deleting list items (for example, deleting the mailto protocol)
- changing attribute values (for example, to make the list of protocol types disappear from the Hyperlink dialog box, change type's visible attribute to "false")

Type List

Determines which protocols a user can assign to a link.

Illustration



Example

```
<command name="cmdhyperlink" enabled="true">

<selections name="type" enabled="false" visible="true" sorted="true">
  <listchoice data="0">file:</listchoice>
  <listchoice data="0">ftp:</listchoice>
  <listchoice data="0">gopher:</listchoice>
  <listchoice data="0" default="true">http:</listchoice>
  <listchoice data="0">https:</listchoice>
  <listchoice data="1">javascript:</listchoice>
  <listchoice data="1">mailto:</listchoice>
  <listchoice data="1">news:</listchoice>
  <listchoice data="0">telnet:</listchoice>
  <listchoice data="0">wais:</listchoice>
</selections>
```

Selection Elements

Element Attribute	Value(s)	Description
name	type	The name of this selection list.
enabled	true (default)	User can select from the list.
	false	User cannot select from the list; selections are grayed out.
visible	true (default)	List is visible.
	false	List is not visible.
listchoice/ data	0	Protocol requires double slash marks (//). For example, <code>http://www.yoursite.com</code> .
	1	Protocol does not require double slash marks (//). For example, <code>mailto:you@email.com</code> .
listchoice/ default	true	This choice is the default type. Note: <code>http:</code> is the default type if no value is specified.
	false (default)	This choice is not the default type.
listchoice/ text value	Any valid protocol (including the colon). Typically, one of the following: <code>file:, ftp:, gopher:, http:, https:, javascript:, mailto:, news:, telnet:, wais:</code>	The internet protocols from which the user can choose.

Target Frame List

Determines target window choices.

Illustration



Example

```
<selections name="target" enabled="true" visible="false" sorted="true">
    <listchoice value="main">Main Frame</listchoice>
    <listchoice value="_blank" localeRef="hypTargB"></listchoice>
    <listchoice value="_self" localeRef="hypTargS" default="true"></listchoice>
    <listchoice value="_parent" localeRef="hypTargP"></listchoice>
    <listchoice value="_top" localeRef="hypTargT"></listchoice>
</selections>
```

Selection Elements

Element Attribute	Value(s)	Description
name	target	The name of this selection list.
enabled	true (default)	User can select from the Target Frame list.
	false	User cannot select from the Target Frame list; selections are grayed out.
visible	true (default)	Target Frame list is visible.
	false	Target Frame list is not visible.
listchoice/value	Any valid frame name or one of the following special names: _blank, _self, _parent, _top	The list of target window types from which the user can choose.
listchoice/localeRef	refID	A code to translate this element within the localization files.
listchoice/default	true	This choice is the default type.
	false (default)	This choice is not the default type.
listchoice/<display text>		Text to appear in the target list if no localeRef is found.

Quick Link List

Populates the “Quick Link” list, the list of URLs or other web destinations to which users will typically want to create jumps.

Illustration



Example

```
<selections name="quicklink" visible="true" bookmarks="true" listtop="false">
  <listchoice href="http://www.ektron.com" target="_blank">Ektron Home Page</listchoice>
  <listchoice href="http://www.ektron.com/support">Ektron Technical Support</listchoice>
</selections>
```

Selection Elements

Element Attribute	Value(s)	Description
name	quicklink	The name of this selection list.
visible	true (default)	The Quick Links list is visible.
	false	The Quick Links list is not visible.
bookmarks	true (default)	Bookmarks on this page appear in the Quick Links list.
	false	Bookmarks on this page do not appear in the Quick Links list.
listtop	true (default)	The "Top" bookmark appears in the Quick Links list.
	false	The "Top" bookmark does not appear in the Quick Links list.
listchoice/ href		The URL of a destination to which the user clicking this link is brought.
listchoice/ target	Any valid frame name or one of the following special names: _blank, _self, _parent, _top	Target window (frame name). If you specify a target frame, and the user is allowed to select a target frame (at the Target Frame field), the user's choice will override this value.
listchoice/ localeRef	refID	A code to translate this element within the localization files (typically not used).
listchoice/ <display text>		Text that describes the destination in the Quick Links list.

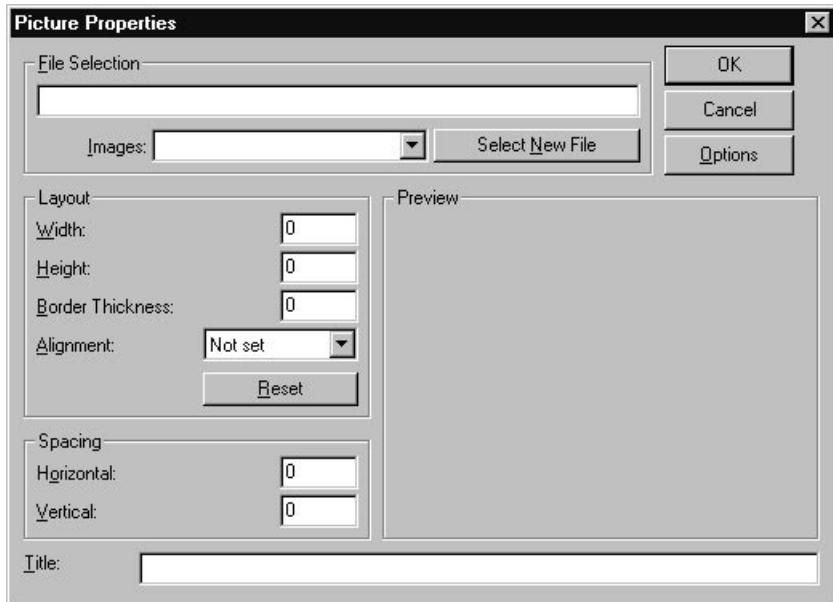
Managing Image Selection

How Image Selection Works

Note

This section assumes that you have not edited the commands in the `mediafiles` feature of the XML configuration data.

1. The user clicks the Insert Image button () , which executes the XML configuration data's `cmdmfumedia` command.
2. The `cmdmfumedia` command calls the `eWebEditProMediaSelection` function in the `ewebeditpromedia.js` file. That function displays the Picture Properties dialog box.



3. The user clicks the **Select New File** button.
4. The editor checks the value of the `type` attribute of the `mediafiles` feature in the XML configuration data.

-
- If you set the value to **FTP**, you need to set up image selection via FTP. (See “For reference, the following illustrates the mediafiles section of the XML configuration data.” on page 244.)
 - If you set the value to an HTML file pathway, that page is loaded. This option typically displays a screen that prompts the user to select an image. More details about this option are provided in “Examples of Implementing Image Selection” on page 236.
5. The Picture Properties dialog box reappears with the selected image. The user can change the image properties if desired.
 6. When the user clicks **OK**, the image is inserted into the content.

Organization of the Image Selection Documentation

The rest of this chapter describes the various aspects of the image selection feature.

This section	Describes
“Programmatically Accessing MediaFile Properties” on page 271	Customizing the external media file selection process.
“Examples of Implementing Image Selection” on page 236	Examples of how to create the image selection screen.
“Implementing Image Upload” on page 250	How to implement media upload under different environments.
“The Mediafiles Feature of the XML Configuration Data” on page 256	The elements of the mediafiles feature.
“Image Selection Object Methods and Properties” on page 266	The methods and properties of the Image Selection Object.
“Programmatically Accessing MediaFile Properties” on page 271	How to programmatically access the MediaFile object’s properties.
“Dynamically Selecting Upload Destinations” on page 281	Using scripting to change the image file upload location

Examples of Implementing Image Selection

This section provides four examples of how to create the image selection screen mentioned in Step 4 of “How Image Selection Works” on page 234. This table summarizes the examples.

Example	File Upload?	Upload protocol	Administrator restricts image?
1: No Restrictions, No Saving to Database	no	n/a	no
2: File Size Restriction, No Saving to Database	no	n/a	yes
3: FTP	determined by web master	FTP	yes
4: Database Samples	yes - URL stored in database	HTTP	yes

Example 1: No Restrictions, No Saving to a Database

In this example, the user inserts an image from a remote directory. The image is not uploaded to a database, and no restrictions are imposed on the image.

To incorporate this version of image selection, follow these steps.

1. Within the ewebeditpro2 directory, create an .htm file, for example, imageselection.htm.
2. Within the imageselection.htm file’s head tags, include the ewebeditpro.js file.

```
<script language="JavaScript1.2" src="ewebeditpro.js">  
 </script>
```

(For more information, see “Entry Point for Using External Scripts” on page 273.)

3. Still within the document’s head tags, create an `insertfile` function that calls the standard `insertMediaFile` function. (See “`insertMediaFile`” on page 104.)

NOTE In the following example, the editor name appears as `MyContent1`. Replace this with the name of the editor from which the user is inserting the image. See Also: “Appendix A: Naming the VisualFormat Editor” on page 300.

```
<script language="JavaScript1.2">
<!--
function insertfile()
{
    top.opener.eWebEditPro.instances["MyContent1"].insertMediaFile(txtpath.value,false,"","IMAGE",0,0);
    window.close();
}
-->
</script>
```

Be sure to specify the parameters for `insertMediaFile`.

Parameter	Value in this example
file location	<code>txtpath.value</code>
is the file local?	<code>false</code>
file title	<code>" "</code>
file type	<code>" IMAGE "</code>
width	<code>0</code>
height	<code>0</code>

NOTE By entering zero (0) as the image’s width and height, the administrator is allowing the image to retain its original dimensions. The user can edit these values in the Picture Properties dialog box, which appears when the image is inserted.

(For more information, see “Specifying an Image to Insert” on page 275.)

4. Enter text to prompt the user to specify the path to the image. For example

Enter path to image file:

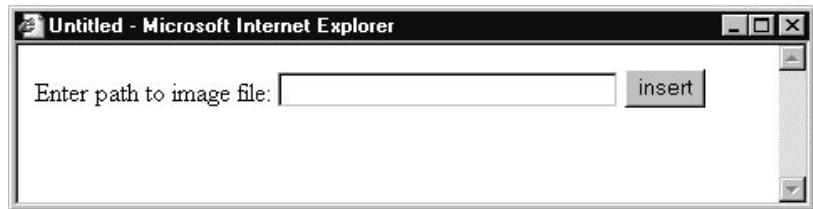
-
5. Create an input field to accept the user's input. For example

```
<input type="text" name="txtpath" size=30 value="">
```
 6. Create a button to invoke the insertfile function.

```
<input type="button" name="btninsert" value="insert" onclick="insertfile()">
```
 7. Open the config.xml file. Within the mediafiles feature, at the transport type attribute, enter the path to the .htm file relative to local host. Place quotes around the path. For example

```
<transport type="/ewebeditpro2/imageselection.htm">
```

As a result, the following screen appears when the user presses the **Select New File** button on the Picture Properties dialog box.



WARNING!

If, while identifying an image, the user enters a pathway in a field used by JavaScript, the user *must* enter two backslash characters wherever they would normally enter one. As an alternative, the JavaScript could convert the backslash characters.

When the user enters a path to an image and clicks the **insert** button, the `insertMediaFile` command passes the image file information to the Picture Properties dialog box.

Below is the. htm file that you would use to implement this version of image selection.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Untitled</title>
<script language="JavaScript1.2" src="ewebeditpro.js">
</script>
<script language="JavaScript1.2">
<!--
function insertfile()
{
    top.opener.eWebEditPro.instances["MyContent1"].insertMediaFile(txtxpath.value,false,"","IMAGE",0,0);
    window.close();
}
-->
</script>
</head>

<body>
enter in a path:
<input type=text name=txtxpath size=30 value="">
<input type=button name=btninsert value="insert" onclick="insertfile()">
</body>
</html>
```

For reference, the following illustrates the `mediafiles` section of the XML configuration data.

```

<mediafiles>
    <command name="cmdmfumedia" style="icon" visible="true">
        <image key="picture"/>
        <caption localeRef="btnTxtrunapp">Image File</caption>
        <toolTipText localeRef="btncrunapp">Image File</toolTipText>
    </command>
    <!-- 0 is unlimited size -->
    <maxsizek>1000</maxsizek>
    <validext>gif,jpg,png,jpeg,jif</validext>
    <mediaconfig enabled="true" allowededit="true"/>
    <!-- If this section is not defined it will default to FTP with no settings -->
    <!-- The attribute 'type' values "ftp" and "file" are handled within the editor. -->
    <!-- The scripting will load the page specified in the type attribute. -->
    <transport enabled="true" type="/ewebeditpro2/ imageselection.htm" confirmation="true"
xfer="binary" pasv="true">
        <!-- Encrypt username and password using Ektron's encrypt.exe program. -->
        <!-- blank for user entry -->
        <username encrypted="true"></username>
        <password encrypted="true"></password>
        <!-- Set to 0 for default port number -->
        <port>0</port>
        <!-- Upload location is: [domain]+[xferdir]+[filename] -->
        <domain></domain> <!-- e.g., ftp.mydomain.com -->
        <!-- Directory transferred into relative to domain -->
        <xferdir src="[ewebeditproPath]/upload"/>
        <!-- Referencing a file through HTTP is: [webroot]+[filename] -->
        <!-- if webroot is blank then it defaults to xferdir value -->
        <webroot src="" />
        <!-- Possible values for resolvepath are: full, host, local, given -->
        <resolvemethod value="local" src="" />
    </transport>
</mediafiles>

```

Example 2: File Size Restriction, No Saving to Database

In this example, the user inserts an image from a remote directory. The web master sets a maximum image size of 100 Kb. If the user tries to insert an image larger than 100 Kb, an error message appears and the insertion is terminated.

NOTE

You can also use the `mediafiles` feature of the XML configuration data to limit the file types that users can insert, using the `validext` attribute. You implement this restriction in the same way you implement maximum file size.

To incorporate this version of image selection, follow these steps.

-
1. Within the `ewebeditpro2` directory, create an `.htm` file, for example, `imageselect_100kb.htm`.
 2. Within the document's head tags, include the `ewbeditpro.js` file.

```
<script language="JavaScript1.2" src="ewbeditpro.js">
</script>
```
 3. Still within the document's head tags, create a JavaScript function (in this example, `sizeisok`) that checks the size of the file selected by the user. If it exceeds 100 Kb, return false; otherwise, return true.

Note that in the example below, the variable `maxsize` refers to the `maxsizek` attribute of the `mediafile` feature in the `config.xml` file. In Step 6, you set the value of the `maxsizek` attribute.

NOTE

In the following example, the editor name appears as `MyContent1`. Replace this with the name of the editor from which the user presses the Insert Picture button. See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.

```
function sizeisok()
{
    var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();
    var maxsize= objmedia.getPropertyInteger("MaxFileSizeK");

    if ((objmedia.FileSize) > maxsize*1024)
    {
        return (false);
    }
    else
    {
        return (true);
    }
}
```

NOTE

If you are using Netscape, you cannot access the ActiveX objects (such as `objmedia.MaxFileSizeK` and `objmedia.FileSize`) directly. Instead, use one of the `getProperty` methods to retrieve these values. (See "Using Netscape to Access Image Properties" on page 272.)

-
4. Create a function (in this example, `insertlocalfile`) that checks the value of the `sizeisok` function.

If the `sizeisok` function returns false, an error message appears ("File is too large."). If the function returns true, the `insertMediaFile` command passes the image file information to the Picture Properties dialog box.

```
function insertlocalfile()
{
    var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();

    objmedia.IsLocal = true;
    objmedia.SrcFileLocationName = txtxpath.value;
    if(sizeisok() == false)
    {
        alert("File is too large.");
    }
    else
    {
        top.opener.eWebEditPro.instances["MyContent1"].insertMediaFile(txtxpath.value,true,"","IMAGE",0,0);
        window.close();
    }
}
```

For a description of the rest of the code in this example, see steps 3 through 6 in “Example 1: No Restrictions, No Saving to a Database” on page 236.

5. Open the config.xml file. Within the `mediafiles` feature, at the `transport type` attribute, enter the path to the .htm file relative to local host. Place quotes around the path. For example

```
<transport type="/ewebeditpro2/imageselect_100kb.htm">
```

6. While in the config.xml file, set the value of the `maxsizek` attribute to 100.

```
<mediafiles>
<maxsizek>100</maxsizek>
```

Below is the entire .htm file that you would use to implement this version of image selection.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!--
How to set up in XML:
<mediafiles>
    .
    <maxsizek>100</maxsizek>
    <transport type="/ewebeditpro2/imageupload_100kb.htm">
    </transport>
</mediafiles>
-->
<html>
<head>
    <title>Untitled</title>
<script language="JavaScript1.2" src="ewebeditpro.js">
</script>
<script language="JavaScript1.2">
<!--
function insertlocalfile()
{
    var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();

    objmedia.IsLocal = true;
    objmedia.SrcFileName = txtxpath.value;
    if(sizeisok() == false)
    {
        alert("File is too large.");
    }
    else
    {
        top.opener.eWebEditPro.instances["MyContent1"].insertMediaFile(txtxpath.value,true,"","IMAGE",0,0);
        window.close();
    }
}
function sizeisok()
{
    var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();
    var maxsize= objmedia.MaxFileSizeK;

    if ((objmedia.FileSize/1024) > maxsize)
    {
        return (false);
    }
    else
    {
        return (true);
    }
}
-->
</script>
</head>

<body>
enter in a path:
<input type=text name="txtxpath" size=30 value="">
<input type=button name="btninsert" value="insert" onclick="insertlocalfile()">
</body>
</html>
```

For reference, the following illustrates the mediafiles section of the XML configuration data.

```
<mediafiles>
    <command name="cmdmfumedia" style="icon" visible="true">
        <image key="picture"/>
        <caption localeRef="btntxtrunapp">Image File</caption>
        <toolTipText localeRef="bttnrunapp">Image File</toolTipText>
    </command>
    <!-- 0 is unlimited size -->
    <maxsizek>100</maxsizek>
    <validext>gif,jpg,png,jpeg,jif</validext>
    <mediaconfig enabled="true" allowededit="true"/>
    <!-- If this section is not defined it will default to FTP with no settings -->
    <!-- The attribute 'type' values "ftp" and "file" are handled within the editor. -->
    <!-- The scripting will load the page specified in the type attribute. -->
    <transport enabled="true" type="/ewebeditpro2/imageselection.htm" confirmation="true"
xfer="binary" pasv="true">
        <!-- Encrypt username and password using Ektron's encrypt.exe program. -->
        <!-- blank for user entry -->
        <username encrypted="true"></username>
        <password encrypted="true"></password>
        <!-- Set to 0 for default port number -->
        <port>0</port>
        <!-- Upload location is: [domain]+[xferdir]+[filename] -->
        <domain></domain> <!-- e.g., ftp.mydomain.com -->
        <!-- Directory transferred into relative to domain -->
        <xferdir src="[eWebEditProPath]/upload"/>
        <!-- Referencing a file through HTTP is: [webroot]+[filename] -->
        <!-- if webroot is blank then it defaults to xferdir value -->
        <webroot src="" />
        <!-- Possible values for resolvemethod are: full, host, local, given -->
        <resolvemethod value="local" src="" />
    </transport>
</mediafiles>
```

Example 3: FTP

You can implement image selection using FTP. To do this, enter **FTP** at the `type` attribute of the `mediafiles` feature of the XML configuration data. Enter the additional FTP information, such as domain, user name, port, and upload location in the `mediafiles` section of the XML configuration data.

Implementing FTP image selection can vary widely, depending on your system. Therefore, the web master should determine how best to implement FTP-based image selection.

Below is an example of how to set up the XML configuration data for a typical FTP site. This example assumes that the FTP site and the Web site share the same physical server.

See also “FTP File Upload” on page 250.

Minimum Configuration Requirements for FTP

These are the minimum configuration requirements if you use the FTP upload mechanism.

- The FTP site and the file’s Web reference site must share the same physical server.
- The FTP server must be configured to allow access to a location that is also accessible through a web browsing mechanism, that is HTTP. If the FTP server is set to start in a directory structure that cannot be reached by a web browser, the uploaded images cannot be displayed.

Server Configuration

Assume that FTP is set up with these parameters.

Domain	ftp.mydomain.com
Physical FTP Root	c:\inetpub\www\ftp
Images reside in	/shared/images
Physical image location	c:\inetpub\www\ftp\shared\images
Connection Port	Standard FTP Port
Data Transfer Style	Binary Data
Connection Mode	Must use passive mode for firewall

Assume that the web site is set up with these parameters.

Domain	www.mydomain.com
Physical WWW Root	c:\inetpub\www
Page Location (Base URL)	/public/pages
Physical Page Location	c:\inetpub\www\public\pages

To implement the above configuration, you would set these values in the `mediafiles` section of the XML configuration data.

```
<transport type="ftp" xfer="binary" pasv="true">
<domain>ftp.mydomain.com</domain>
<xferdir src="/shared/images"/>
<webroot src="http://www.mydomain.com/public/pages"/>
```

Notice that since this example uses the standard FTP port, it does not include the `port` element.

Restriction Settings

To continue with the example, the administrator wants to add these restrictions to any uploaded images.

File Extensions	gif, jpg
Maximum File Size	12K
Login	User must log in to FTP account
File Referencing	All reference paths are relative to the local page

To implement these restrictions, you would set these values in the `mediafiles` section of the XML configuration data.

```
<validext>gif,jpg</validext>
<maxsizek>12</maxsizek>
<username></username>
<password></password>
<resolvemethod value="local"/>
```

User Interface Control

The administrator does not want to let the user review any of the settings. The login dialog must be shown for the user to log in.

```
<mediacconfig enabled="true" allowededit="false"/>
```

FTP Configuration in XML

The sample configuration described above will use this example MediaFiles section of the XML configuration data.

```
<mediafiles>
    <command name="cmdmfumedia" style="icon" visible="true">
        <image key="picture"/>
        <caption localeRef="btnTxtrunapp">Image File</caption>
        <toolTipText localeRef="btncmdfumedia">Image File</toolTipText>
    </command>
    <maxsizek>12</maxsizek>
    <validext>gif,jpg</validext>
    <mediaconfig enabled="true" allowededit="false"/>
    <transport type="ftp" xfer="binary" pasv="true">
        <username></username>
        <password></password>
        <port>0</port>
        <domain>ftp.mydomain.com</domain>
        <xferdir src="/shared/images"/>
        <webroot src="http://www.mydomain.com/public/pages"/>
        <resolvemethod value="local"/>
    </transport>
</mediafiles >
```

Example 4: Database Samples

When you install VisualFormat, you have an option to install database samples for your platform. For example, if you are running ASP, you can install ASP database samples.

See also “ASP” on page 252

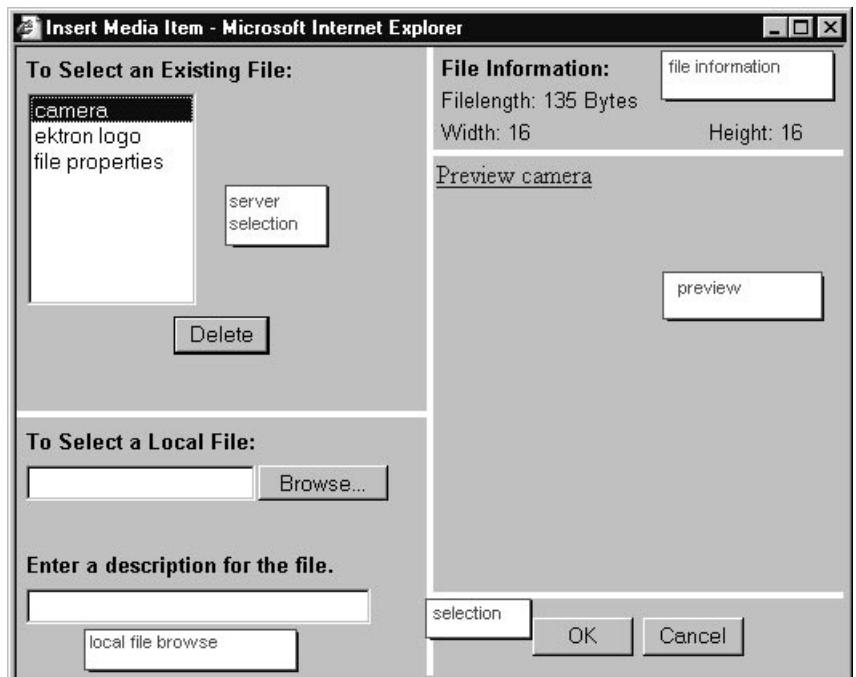
If you install database samples, a sample image selection screen is provided. (Where the image selection screen fits into the workflow of selecting an image is explained in Step 4 of “How Image Selection Works” on page 234.)

The sample screen lets the user select images from local directories or a server, upload files to a server, and preview an image before returning to the Picture Properties dialog box.

You can use the sample image selection screen as is, or modify it as needed for your users.

Below is the ASP sample image selection screen, with callout boxes to label the areas of the

screen.



The following table describes the files that make up the ASP database sample. Samples for other platforms use essentially the same files -- only the file extensions are different.

File Name	Frame Name	Function - Allows the user to	Operation
medialist.asp	server selection	Select a file that resides on the server.	Retrieves titles of all media files in the database, then builds an option list box for displaying the titles. The user can highlight the desired title. When the user highlights a title, the preview frame, the local file browse frame, the file information frame, and the selection frame are updated to reflect the selection.
mediauploader.asp	local file browse	Choose a file from the local system. The local file is uploaded before it is inserted into the editor.	Lets the user choose a local file and assign it a title. When the user enters a local file, the server selection frame, the file information frame, the preview frame, and the selection frame are updated to reflect the selection.
mediainformation.asp	file information	View file information, including its length in bytes and, if the file is an image, its width and height in pixels.	Displays information about file the user selected, whether the file is server-based or local.
mediapreview.asp	preview	Preview the selected file.	Lets the user preview the highlighted file before selecting it.
mediainsert.asp	selection	Select a file. If a local file is selected, the file is uploaded before it is inserted into the editor.	Allows the user to select a server file or a local file. It also ensures that a title has been entered if the user selects a local file.

Implementing Image Upload

This section describes the following methods and options for enabling users to upload images and other files to your web server.

- FTP
- HTTP
 - ASP
 - ColdFusion
 - other web servers

Security issues surrounding each approach are explained.

FTP File Upload

You can use FTP (file transfer protocol) to copy files from the user's (or client) computer to the web server. The web server must have an FTP server to establish a connection and receive a file from the client computer. Many server operating systems provide an FTP server. Commercial FTP server software is also available.

VisualFormat can notify the server when a file is uploaded via FTP. This capability allows the server to update a database with the list of uploaded files.

To enable this notification, implement the `eWebEditProUploadNotification` function in JavaScript. This function opens a dynamic web page and passes file information to the server, typically through URL parameters.

Security with FTP

Usually, you have an FTP account with a user name and password. When uploading files through VisualFormat, your FTP user name and password must be specified. To keep them secret, use Ektron's encryption program to scramble your user name and password.

Enter the user name and password in the `username` and `password` elements of the `mediafiles` feature of the XML configuration data. An example appears below.

```
<mediafiles>
.
.
.
<transport enabled="true" type="ftp" confirmation="true" xfer="binary" pasv="true">
    <!-- Encrypt username and password using Ektron's encrypt.exe program. -->
    <!-- blank for user entry -->
    <username encrypted="true">zVQjUOPG</username>
    <password encrypted="true">uDekdcUF</password>
.
.
.
```

You may download Ektron's encryption program and view the Encryption User's Guide from Ektron's web site, www.ektron.com.

HTTP File Upload

You can use HTTP (the same protocol that displays a web page) to upload image files from a user's computer to a web server. All web servers support HTTP, but they usually require additional software to receive files from a client computer. Many web application servers, such as ASP and ColdFusion, provide functions to write files to the web server's file system.

NOTE

If a user deletes an image from the images list, the image is removed from the database but not from the physical directory on the server.

Overview

HTTP image upload with VisualFormat uses standard web pages. You can write your own or use the samples provided with VisualFormat.

Typically, web pages that upload images or other files include the elements shown below. Note that

-
- the `enctype` must be "**multipart/form-data**"
 - you must specify an action page
 - the input type of "file" displays a text box for the file name and a **Browse** button that lets the user select a file to upload

```
<form name="MyFormName" method="post" action="MyActionPage.xyz"
      enctype="multipart/form-data" OnSubmit="return MyValidateFormData()">
  ...
<input type="file" name="MyUploadFile" size="20" maxlength="256" align="MIDDLE">
  ...
</form>
```

ASP

Microsoft Active Server Pages (ASP) include the ability to write text files to the file system, but do not have the native ability to write binary files. Since images (GIF and JPG) and other files (such as, audio, video, and Microsoft Office documents) are binary, an additional component is required.

Older versions of VisualFormat on ASP used a proprietary method to upload image files via HTTP. A server-side COM DLL, EkFileIO.DLL, was required to save the image file.

With VisualFormat 2.0, image upload uses standard multipart form data to upload the file. As a result, you can use any commercially-available file upload software for ASP. A popular file upload package for ASP is FileUp, available from SoftArtisans at <http://www.softartisans.com/softartisans/saf.html>.

Ektron still provides a server-side COM DLL, EktronFileIO.DLL (note the name change), for file upload support on a Windows NT 4 or Windows 2000 server.

EktronFileIO.dll

EktronFileIO.dll is a COM object that retrieves a file from multipart form data that has been submitted to the server. It then writes the file to the server's file system.

The COM object is created by the action page that is opened when the form is submitted. The ASP database sample, supplied with VisualFormat 2.0, includes EktronFileIO.dll and an action page, medianotification.asp, to receive uploaded files. In it, you will see

the object created using
CreateObject("EktronFileIO.EkFile").

Registering EktronFileIO.dll

EktronFileIO.dll adds information to the Windows registry that allows the ASP page to create the COM object. As a result, you must register EktronFileIO.dll on the web server before you can use it. If you ran the Windows installation and installed the ASP sample, the EktronFileIO.dll is already registered.

If you need to register EktronFileIO, open a command prompt and run regsvr32. By default, the EktronFileIO.dll is located in the /ewebeditpro2/samples/asp/database directory under the web root, but it can reside anywhere on the server.

Here is the code you would enter to register EktronFileIO.dll if it is in the default directory.

```
cd \inetpub\wwwroot\ewebeditpro2\samples\asp\database  
regsvr32 EktronFileIO.dll
```

Licensed owners of VisualFormat 2.0 may download EktronFileIO.dll onto their Windows web server.

Security with ASP

The image selection page with the Browse button should validate the file extension to upload. Security should also be in the ASP page that is the form's action page.

The ASP page should check the file type and only accept files that are safe, such as image files with extensions: gif, jpg, or png (see “validext” on page 256). You may also want to allow document files, such as, doc and pdf extensions, or media files, such as wav, ram, and asf.

You should not allow ASP or HTML files to be uploaded; a malicious person could gain control over the web server and cause damage.

For best security, only allow authorized users to access a page with VisualFormat on it. Windows Server provides a user authentication capability.

Alternatively, you could use FTP, which is protected with a password. Image upload can be disabled altogether on VisualFormat if needed.

The ASP database sample supplied with VisualFormat 2.0 includes an action page, medianotification.asp, to receive uploaded files.

ColdFusion

Macromedia/Allaire ColdFusion server has a CFFILE feature that enables you to save files to the server's file system. See the ColdFusion server documentation for details on CFFILE.

The ColdFusion database sample, supplied with VisualFormat 2.0, includes an action page (medianotification.cfm) and a custom tag file (ewebeditproupploadfile.cfm) to receive uploaded files. In it, you see the `<cffile action="UPLOAD" ...>` tag.

Security with ColdFusion

The image selection page with the Browse button should validate the file extension to be uploaded. Security should also be in the ColdFusion page that is the form's action page. The ColdFusion page should check the file type and only accept files that are safe, such as image files with extensions: gif, jpg, or png. You may also want to allow document files, such as, doc and pdf extensions, or media files, like, wav, ram, and asf (see "validext" on page 256).

You should not allow CFM or HTML files to be uploaded; a malicious person could gain control over the web server and cause damage.

For more on security issues with CFFILE, see the ColdFusion article "Top Five ColdFusion Security Issues" (<http://www.allaire.com/Handlers/index.cfm?ID=19491&Method=Full&Cache=Off>).

For best security, you should only allow authorized users to access a page with VisualFormat on it. Most web servers provide user authentication. Alternately, you could use FTP, which is protected with a password. If needed, you can disable Image Upload. The ColdFusion administrator can enable or disable the CFFILE tag.

Other Web Servers

Your web application server must support file upload and provide an ability to write binary files to the server's file system. Files are uploaded using HTTP in a web page form using multipart form data. Check your documentation for instructions. Third party software may also be available.

Security

The image selection page with the Browse button should validate the file extension to upload. Security should also be in the dynamic web page that is the form's action page.

The page should check the file type and only accept files that are safe, such as image files with extensions: gif, jpg, or png. You may also want to allow document files, such as, doc and pdf extensions, or media files, like, wav, ram, and asf (see "validext" on page 256).

You should not allow dynamic pages and HTML files to be uploaded; a malicious person could gain control over the web server and cause damage.

For best security, only allow authorized users to access a page with VisualFormat on it. Most web servers provide user authentication.

Alternatively, you could use FTP, which is protected with a password. Image upload can be disabled altogether on VisualFormat, if needed.

The Mediafiles Feature of the XML Configuration Data

This section describes the elements of the `mediafiles` feature of the XML configuration data.

mediafiles

Description

Defines the configuration options for the `mediafiles` feature.

Location

```
<config>
  <features>
    <mediafiles>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Defines whether the feature is enabled. If set to “False”, the feature is not available to the user.

validext

Description

Specifies the valid extensions allowed for upload.

The editor removes wildcard characters and spaces from the list. So, for example, gif, .gif and *.gif are treated the same.

Location

```
<config>
  <features>
    <mediafiles>
      <validext>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	A comma delimited list that specifies file extensions allowed for upload.

maxsizek

Description

Specifies the maximum file size allowed for upload. This is an integer value that specifies the number of Kilobytes in the file.

Location

```
<config>
  <features>
    <mediafiles>
      <maxsizek>
```

Attributes

Name	Attribute Type	Default	Description
#text	Integer	0	This is interpreted as an integer value specifying the number of kilobytes in the file. If the value is zero, the file has no size limit.

transport

Description

Defines the mechanism used to select and upload media files.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
```

Attributes

Name	Attribute Type	Default	Description
Enabled	Boolean	true	If set to " False ", all transport values are blank.
Type	String	"FTP"	Specifies the upload mechanism to use. Values handled internally are " FTP " and " FILE ". Any other values are passed to the client application or script. The case is maintained.
Pasv	Boolean	True	Specifies whether the passive bit is set for FTP.
Allowupload	Boolean	True	Specifies whether a user can upload files. If true, users should only be allowed to select files already on the server. The upload mechanism (such as an external page) must use this value to prevent users from uploading files.
Xfer	String	"binary"	Specifies the low level transfer option for FTP.

username

Description

Provides the user name for gaining access to the server. This value can be encrypted using the Ektron encryption software. Decryption is done using the licensing key provided to the editor.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <username>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The user name. Since not all external mechanisms required login access for uploading, this is optional.
Encrypted	Boolean	True	If "true", the value contained in #text is encrypted and will be decrypted before it is used.

password

Description

Provides the password for gaining access to the server. This value can be encrypted using the Ektron encryption software. Decryption is done using the licensing key provided to the editor.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <password>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The password. Since not all external mechanisms required login access for uploading, this is optional.
Encrypted	Boolean	True	If "true," the value contained in #text is encrypted and will be decrypted before it is used.

proxyserver

Description

Specifies the proxy server to use. Normally, this value is for FTP only.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <proxyserver>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The server name or TCP/IP address. Proxy servers are not always required.

domain

Description

The domain name for the connection.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <domain>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The domain name or TCP/IP address. If blank, the editor will try to determine the current domain. External mechanisms do not require a domain name, but can use one if needed.

xferdir

Description

The destination directory on the server for the upload. When referenced from FTP, this is a different location than when referenced from the web. For example, when referenced from the web, the path might be ..\dir1\dir2\image.gif or, as a full path, /topdir/ftp/dir1/dir2. In contrast, when referenced from FTP, the path might be /dir1/dir2/image.gif.

For ASP, Cold Fusion, JSP, and other external mechanisms, the references are the same.

If the upload location and the reference location are the same, leave the `webroot` element blank. It will inherit the value from `xferdir`.

See also: “BaseURL” on page 116.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <xferdir>
```

Attributes

Name	Attribute Type	Default	Description
src	String	""	The destination directory for uploaded files. The case is maintained.

webroot

Description

Specifies the path to use when referencing an uploaded file.

If the server/domain is different from the upload server/domain, this value must contain the new domain, such as:

`http://www.yahoo.com/images`

If the webroot has no value, it inherits the value of the `xferdir` element.

If the Web reference domain is different from the transfer domain, the domain name must be included in the webroot element.

NOTE

If you enter the domain in the webroot element, you must include the protocol. For example `HTTP://www.mydomain.com/public/pages`.

See also: “BaseURL” on page 116.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <webroot>
```

Attributes

Name	Attribute Type	Default	Description
src	String	""	The reference location for uploaded files. The case is maintained. If not included or blank, the value of xferdir is used.

defsource

This specifies the location where a user is placed when browsing for a file on a local system. The path given can be anywhere on the local drive or network server.

Normally, this value is used by FTP upload to help select a local file. An external selection mechanism can also use this value to specify where to retrieve a list of files.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <defsource>
```

Attributes

Name	Attribute Type	Default	Description
src	String	""	The location to start browsing for files to upload. The case is maintained.

port

Specifies which port to use for any file transfers. This value is only required if a non-standard port is used. If the value is zero or is not included, the editor will determine the correct port to use.

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <port>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	0	The port to use for file transfers. If not given or set to zero, the editor determines which port to use based on the selected protocol.

resolvemethod

Description

Defines how to resolve file paths. Paths are resolved relative to the base URL (that is, the current page location).

Method	Resolves	Example
FULL	All path names to include the protocol, domain, and full path. This method ensures that paths are correct regardless of where they are referenced from.	http://www.yahoo.com/pages/images/me.gif
HOST	Relative to the root of the host server. This method lets you move directory structures to a publishing server without having to change any paths.	/pages/images/me.gif
LOCAL	Relative to the current location. This method lets you move directory structures up and down within file systems as well as to other servers.	./images/me.gif
GIVEN	To a given future location. This method resolves the paths to the location where the files will be moved. The resolved path is similar to local.	../publish/images/me.gif

Location

```
<config>
  <features>
    <mediafiles>
      <transport>
        <resolvemethod>
```

Attributes

Name	Attribute Type	Default	Description
Value	String	LOCAL	The resolve method to use when resolving paths. The valid values are FULL , HOST , LOCAL , and GIVEN .
Src		""	The path to use for the GIVEN resolution. Case is maintained.
Allowoverride	Boolean	False	If set to “True”, the user can disable the path resolution mechanism. If disabled, paths entered by the user are not modified.
Resolve	Boolean	True	If “True”, the path resolution mechanism is enabled and will resolve paths according to the specified mechanism. If disabled, paths entered by the user are not modified.

Image Selection Object Methods and Properties

Image Selection Object Methods

Method	Return Type	Description
FileExistsLocally()	Boolean	Uses the value given to SrcFilePath to determine if the file exists on the local system. This can be used for error checking: if the user types in a bad path, this method can detect it.

Image Selection Object Properties

The properties contain information about the file, the source location, and the destination.

The object automatically parses the path and uses the values from some of the properties to determine a transfer destination path and a reference path. Initial values for several of these parameters are specified in the `mediafiles` feature of the XML configuration data.

Property	Type	Description
Alignment	String	<p>The image's alignment on the page.</p> <ul style="list-style-type: none">• left• right• top• middle• bottom• AbsMiddle• AbsBottom <p>The user can edit this value in the Picture Properties dialog box.</p>

Property	Type	Description
AllowSubDirectories	Boolean	Determines whether or not a user can select sub-directories. If false, the user cannot. Currently set but not implemented.
AllowUploads	Boolean	If true, the user can upload files from the local PC to the server. If false, the user can only insert files that reside on the server. Note: It is up to the upload mechanism to use this value. For FTP, if this value is false, FTP does not let the user upload files. It only lists the available files. The ASP and ColdFusion samples work the same way. If the value is false, the upload frame is blank.
BaseUrl	String	The base URL value set in the editor. This is a friend property. It should be set by a routine that knows the base URL.
BorderSize	Integer	The size of the image's border in pixels. The user can edit this value in the Picture Properties dialog box.
ConfirmBeforeUpload	Boolean	If true, the user is asked to proceed with the upload after selecting an image. If false, the upload occurs without user confirmation.
DefDestinationDir	String	The destination path to where the image will be placed. This is the same as the TransferRoot.
DefSourceDir	String	The initial directory that appears when the user is selecting a local file.
Domain	String	The domain name of the upload server. This is mainly for use with FTP, but may also be important for other upload mechanisms.
FileSize	Long	The size of the image file in bytes. This value is set when the user selects a local file.
FileTitle	String	The title of the file. This is not the file name but a descriptive title that users assign after selecting the file. It is used as the image's alt text.
FileType	String	The type of file. These are the choices. <ul style="list-style-type: none"> ● bitmap ● video ● audio ● document ● other
FWLoginName	String	User's login name for the firewall. Not currently used.
FWPassword	String	User's password for the firewall. Not currently used.
FWPort	Integer	The firewall port to use for any transfer. If this value is zero (0), the transfer mechanism determines the port. Not currently used.

Property	Type	Description
FWProxyServer	String	Firewall proxy server. Not currently used.
FWUse	Boolean	If true, a firewall mechanism is used. Not currently used.
FWUsePassV	Boolean	If true, PASV mode FTP is enabled.
HandledInternally	Boolean	The upload has already been handled internally. If true, the upload is skipped, and only the notification is done.
HorizontalSpacing	Integer	Horizontal spacing attribute to use in the HTML. The user can edit this value in the Picture Properties dialog box.
ImageHeight	Integer	The height of the image. This value is set when an image is selected. See also ShowHeight.
ImageWidth	Integer	The width of the image. This value is set when an image is selected. This is not a rendered size, but the actual size of the image. See also ShowWidth.
IsLocal	Boolean	Set this to true if a local file will be placed into the SrcFileName property. The object processes the path information differently for local files. If this value is not set, the object resolves the source location to a remote path, and upload is not possible.
LoginName	String	The login name of the user uploading the image. This may be encrypted in the XML configuration data.
MaxFileSizeK	Integer	The maximum size in kilobytes of an image to be uploaded. A value of zero (0) means no size limit.
NeedConnection	Boolean	A read-only property that determines if a connection is necessary with the current upload method.
Password	String	The password of the user uploading the image. This may be encrypted in the XML configuration data.
Port	Integer	The port to use for uploads. If zero (0), the file's upload type determines the port.
ProxyServer	String	The name of the proxy server to use with uploads. This property is not required. Proxy servers are primarily used with FTP.
RemotePathFileName	String	The remote path and name of the currently selected file. This path may have been generated using the path parameters when a local file is entered into SrcFileName. The application can also set a remote path and name to override the generated one.

Property	Type	Description
ResolveMethod	String	The method by which the image source path is resolved. The choices are: FULL - fully qualified to server HOST - relative to host LOCAL - relative to page GIVEN - relative to given location - WebRoot
ResolvePath	String	The path used to resolve an image path when GIVEN is used as the resolution method. It defaults to the WebRoot, since files are uploaded there.
ShowHeight	Integer	The height attribute of the HTML image tag. Enter a value here if you want to determine the image's height, regardless of its actual size (which is stored in the ImageHeight property). This value defaults to the ImageHeight property value. The user can edit this value in the Picture Properties dialog box.
ShowWidth	Integer	The width attribute for the HTML image tag. Enter a value here if you want to determine the image's width, regardless of its actual size (which is stored in the ImageWidth property). This value defaults to the ImageWidth property value. The user can edit this value in the Picture Properties dialog box.
SrcFileLocationName	String	The full location of the source file. This includes the server, if applicable, and the path and file name with extension.
TransferMethod	String	The name of the upload method used if the ProvideMediaFile method is called. The value of this parameter determines what the upload mechanism should do. The string can be anything from a key word to a URL. If it is not an internal value, a script must interpret it. The internal values are FTP and FILE. For more information on FILE, see "Setting up an Image Repository" on page 278.
TransferRoot	String	The same as the DefDestinationDir.
UsePassV	Boolean	If true, FTP works in passive mode.
ValidConnection	Boolean	If true, the system has made a valid connection with the current connection parameters.
ValidExtensions	String	The file extensions of images that can be uploaded, entered as a comma-delimited string. For example "gif,tif,jpg"
VerticalSpacing	Integer	The value of the vertical spacing attribute of the HTML image tag. The user can edit this value in the Picture Properties dialog box.

Property	Type	Description
WebPathName	String	The web accessible name of the specified file. The name is resolved using the rules assigned to the ResolveMethod value specified. For example <code>http://www.ektron.com/images/me.gif</code> .
WebRoot	String	The base location for accessing uploaded images from a web page. For example <code>http://www.ektron.com/images</code> .

Programmatically Accessing MediaFile Properties

The MediaFile object provides access to image properties relating to the file and the upload process. Values set for these properties affect the operation of the editor.

“Image Selection Object Properties” on page 266 lists the properties. You can set default values for most properties in the XML configuration data.

This section provides the following topics, which explain how to *programmatically* access the image properties under various circumstances.

- Accessing the Media File Object
- Using Netscape to Access Image Properties
- The Entry Point for Using External Scripts
- Setting External Page Parameters
- Changing the Transfer Method on the Fly
- Specifying an Image to Insert
- Modifying the Upload Directory

Accessing the Media File Object

You gain access to the media file object properties programmatically via the `MediaFile` method in the `VisualFormat` control.

```
Function getValidExtensions(seditorname)
{
    var objMedia = top.opener.eWebEditPro[sEditorName].MediaFile();

    return(objMedia.getPropertyString("ValidExtensions"));
}
```

See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.

Using Netscape to Access Image Properties

Within Netscape, the Esker ActiveX plug-in converts the ActiveX control to a plug-in that Netscape can interpret. As a result, you cannot access MediaFile object image properties directly. Instead, use the `getProperty` and `setProperty` methods listed below.

```
setProperty(strName, strValue)
getProperty(strName) as Object
getPropertyInteger(strName) as Integer
getPropertyString(strName) as String
getPropertyBoolean(strName) as Boolean
```

Below are examples of their usage.

```
bIn = eWebEditPro[sEditorName].MediaFile().getPropertyBoolean("HandledInternally");
bUpload = objMedia.getPropertyBoolean("AllowUpload");
sExt = objMedia.getPropertyString("ValidExtensions")
iSz = top.opener.eWebEditPro[sEditorName].MediaFile().getPropertyInteger("MaxFileSizeK");
objMedia.setProperty("SrcFileLocationName", cStr);
eWebEditPro[sEditor].MediaFile().setProperty("TransferMethod", "mediamanager.cfm");
```

Similar property access is done within Java applications. The Java Bean file provides the functionality for accessing properties.

Entry Point for Using External Scripts

The ewebeditpromedia.js file contains the entry point for external scripting of image selection and upload. Its contents are below.

```
// Copyright 2000-2001, Ektron, Inc.  
// Revision Date: 2001-04-03  
  
// Media Upload Functionality  
// Modify this file to customize file upload capability.  
  
function eWebEditProMediaSelection(sEditorName)  
{  
    // The transfer method specifies what to load for the transfer.  
    var objMedia = eWebEditPro[sEditorName].MediaFile();  
    var XferMethod = objMedia.getPropertyString("TransferMethod");  
    var sPageLoad = escape(XferMethod) + '?editorname=' + escape(sEditorName) +  
    '&upload=' + escape(objMedia.getPropertyBoolean("AllowUpload"));  
  
    if(XferMethod != "")  
    {  
        window.open(sPageLoad, 'Images',  
        "scrollbars,resizable,width=640,height=480");  
    }  
    else  
    {  
        alert('The Transfer Method value is empty. Please specify either "FTP" or  
        a site address that will handle the file selection.');//  
    }  
}
```

The page value is specified in XML like this.

```
<features>  
    . . .  
    <mediafiles>  
        . . .  
            <transport type="samples/asp/database/mediamanager.asp">  
                . . .  
            </transport>  
        </mediafiles>  
</features>
```

You can also specify the transport type by modifying the TransferMethod property of the MediaFile object. The ASP and ColdFusion samples demonstrate this.

Setting External Page Parameters

External pages can pass two parameters to help process the image request.

- Editor's Name (editorname)
- Upload Access (upload)

The parameters are passed like this.

```
mediamanager.cfm?editorname=MyContent1&upload=true
```

Use the (editorname) parameter to access the editor in scripts. The parameter is the name of the editor that processed the command to bring up the page. The name can be anything. In the sample files provided, the name is MyContent1 or MyContent2.

See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.

The Upload Access (upload) parameter specifies whether the user can upload image files to the server. (See also **AllowUploads** in "Image Selection Object Properties" on page 266.)

Advanced users can specify their own parameters in the XML configuration data or set them in the XferType property of the MediaFile object. Custom parameters *must* appear at the beginning of the parameter list. The two standard parameters are appended to the end of the list.

For example, a user wants to pass the domain name as a parameter. Here is how you would define this in the XML configuration data.

```
<transport type="mymediaupload.cfm?domain=mydomain">
```

Here is how you would define this in the script.

```
var objMedia = top.opener.eWebEditPro.instances[sEditorName].editor.MediaFile();
objMedia.XferType = "mymediaupload.cfm?domain=mydomain";
```

Changing the Transfer Method on the Fly

This example shows how to specify a page while running the script.

```
function initTransferMethod(sEditor)
{
    eWebEditPro[sEditor].MediaFile().setProperty("TransferMethod", "mediamanager.cfm");
}
```

Specifying an Image to Insert

This Java Script example shows how to insert an image that was loaded by an external mechanism.

```
Function useSelectedFile(seditorname, sfilename, stitle)
{
    //This will bring up the properties dialog and have the user confirm the insert.
    top.opener.eWebEditPro.instances[seditorname].insertMediaFile(sfilename, 0,
    stitle, filetype[iloop], 0, 0);
}
```

The insertMediaFile function is defined in the core java script. (See “insertMediaFile” on page 104.)

Below is the code in the Core Javascript.

The script must inform the MediaFile object that the file about to be specified is remote. To do this, set the `IsLocal` property to **false**.

```
function eWebEditProEditor_insertMediaFile(strSrcFileLocation, bLocalFile, strFileType,
strFileType,
nWidth, nHeight)
{
    setTimeout('eWebEditPro.instances["' + this.name + '"].insertMediaFileDeferred("' +
strSrcFileLocation + '", ' + bLocalFile + ', "' + strFileType + '", "' + strFileType + '", ' +
nWidth + ', ' + nHeight + ')', 1);
}

function eWebEditProEditor_insertMediaFileDeferred(strSrcFileLocation,
bLocalFile, strFileType, nWidth, nHeight)
{
    // Place the file information into the media file object.
    // This is used for the insertion of the HTML.
    var objMedia = this.editor.MediaFile();

    objMedia.setProperty("IsLocal", bLocalFile);
    objMedia.setProperty("SrcFileName", strSrcFileLocation);
    objMedia.setProperty("FileTitle", strFileType);
    objMedia.setProperty("FileType", strFileType);
    objMedia.setProperty("ImageWidth", nWidth);
    objMedia.setProperty("ImageHeight", nHeight);

    this.editor.ExecCommand("cmdmfuinsert", strSrcFileLocation, bLocalFile);
}
```

This example also does not specify a width and height. If they are not specified, the properties dialog box offers to the user the ability to retrieve the file and determine the dimensions.

Modifying the Upload Directory

Here is an example of changing the upload and reference directory while executing a script.

For server side functionality, such as ASP, JSP, ColdFusion, and PHP, the transfer directory and the reference directory should be set the same. Other upload functionality, such as FTP, may have these as two different directories. This sample assumes server side functionality such as ASP or ColdFusion.

```
function SetTransferDirectory(seditorname, spathname)
{
    // This sets the transfer directory for the named editor.
```

```
top.opener.eWebEditPro[seditorname].MediaFile().setProperty("TransferRoot",  
spathname);  
  
// Since the upload and web reference are the same, we should also  
// ensure that the reference path is the same.  
top.opener.eWebEditPro[seditorname].MediaFile().setProperty("WebRoot", spathname);  
}
```

See Also: “Appendix A: Naming the VisualFormat Editor” on page 300.

Setting up an Image Repository

VisualFormat lets you set up an image repository folder on an intranet. Keeping all images in a central location makes it easy for users to select an image and insert it into their web content.

The Image Repository Folder

You should create an image repository folder on a server that is accessible to client PCs, either through a UNC path or a mapped drive. (You can test this through Windows Explorer).

Next, specify the pathway to that folder in the `xferdir` attribute of the transport feature within the XML configuration data. You must enter the full path to the folder -- relative paths are not allowed. Below are two examples:

```
<xferdir src="\\imageserver\GIFs" />
<xferdir src="M:\images\GIFS" />
```

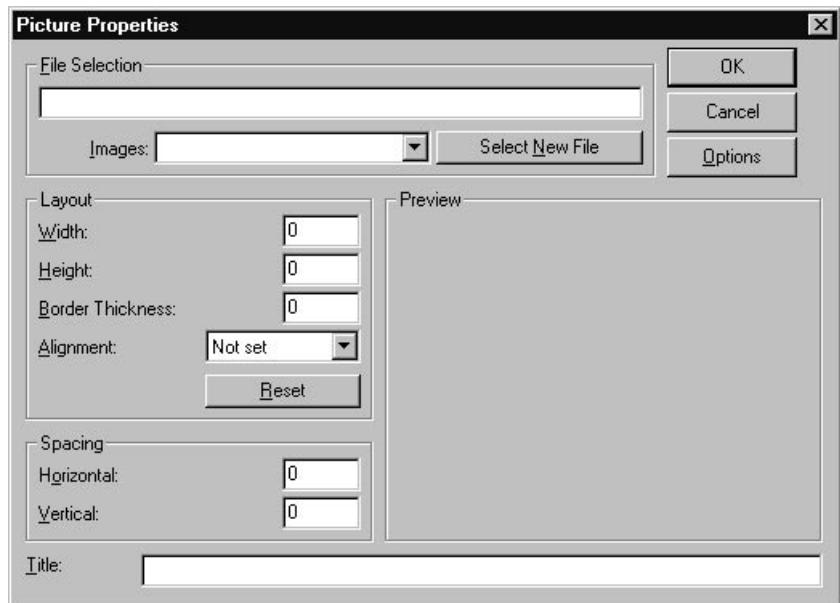
Also, set the `type` attribute to “`file`”.

```
<transport enabled="true" type="file" xfer="binary" pasv="true">
```

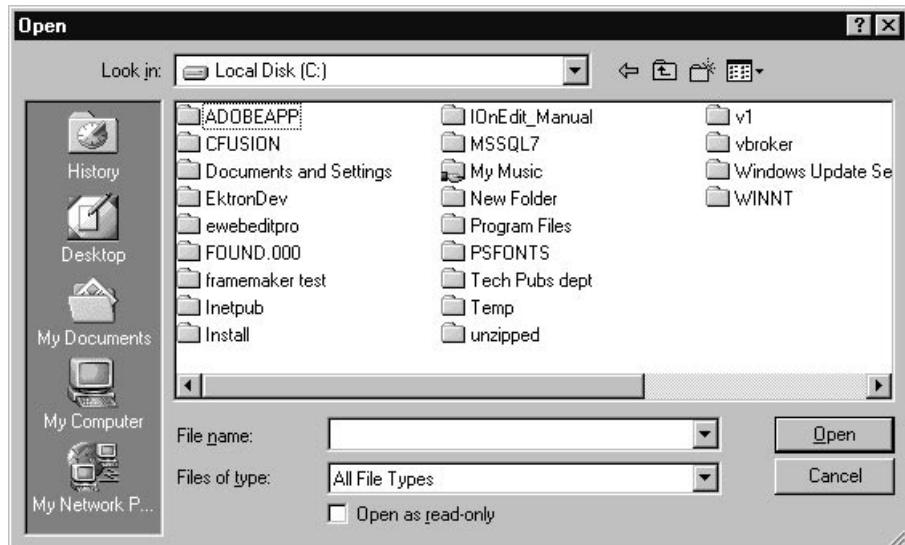
`xferdir` and `type` are the only elements under `transport` that you need to define. You do not need to define any other elements, such as `webroot`, `password`, `domain`, or `resolvemethod`.

Inserting an Image into a Web Page

To insert an image, the user clicks the insert picture button (



Here, the user clicks **Select New File** and navigates to the folder containing the image, using the standard Windows file selection dialog box.



If a user selects an image from the image repository folder, VisualFormat references the image in the web document.

Only files with the following extensions can be inserted with the Picture Properties dialog box: gif, jpg, jpeg, png, tif, bmp, tga, emf, wmf, img, pic, pcx.

If a user selects an image that is not in the image repository folder, VisualFormat copies the image to that folder. The web document references the image from the folder. From then on, all users with access to the folder can insert the image.

NOTE You cannot copy a file to the image repository folder if a file of the same name already resides there.

When the user inserts an image, the full path to the image is saved with the web document. For example, if you insert an image named button.gif into your web document, the HTML code for that line might look like this.

```
<src="file:///M:/images/GIFS/button.gif" />
```

No other resolution options are available to the “**file**” upload type.

Example

Below is a full example of the `mediafiles` section that defines the file upload method. The `transport` element defines the upload method. All other sections are for the general feature definition.

```
<mediafiles>
    <transport type="FILE">
        <xferdir src="M:\images\GIFS"/>
    </transport>
    <command name="cmdmfumedia" enabled="true">
        <image key="picture"/>
        <caption localeRef="btnCapPic"/>
        <tooltiptext localeRef="btnTipPic"/>
    </command>
    <maxsizek>0</maxsizek>
    <validext>gif,jpg,png,jpeg,jif</validext>
    <mediaconfig enabled="true" allowedit="true"/>
</mediafiles>
```

Dynamically Selecting Upload Destinations

Summary

When an image file is uploaded, it is moved to an upload directory defined in the XML configuration data. Often, you need to change the directory, depending on user interaction or other changing conditions. You can use client scripting to modify the upload directory at almost any time.

This section describes how to use scripting to change the image file upload location. The examples in this section illustrate how to change the image upload directory.

In the first example, an external upload mechanism is assigned. In web browsers, this is an external page that contains the functionality for uploading files. The ASP sample installed with VisualFormat is used here.

In the second example, the upload directory is assigned and modified. The user is presented with three choices of a directory and can change it at any time. (Three choices is not a limit of the editor—it is just a number used in this example.) The image files are uploaded, and the path information is stored in the database. The user can then select and view the files anywhere in the editor.

NOTE

The upload location, as well as other settings, can only be changed during or after the “ready” notification. This notification generates a call to the eWebEditProReady function, which the site administrator creates. If changes to the upload location are made before this notification, the settings are replaced by the values in the XML configuration data.

Implementing Image Upload

Background

The uploading of image files and other files is controlled through the XML configuration data. This data includes an upload mechanism, an upload destination directory, referencing information, and other information.

Here is an example configuration for the internal FTP image upload mechanism.

```
<mediafiles>
    <transport type="FTP">
        <domain>ftp.mydomain.com</domain>
        <xferdir src="/pages/[eWebEditProPath]/upload"/>
        <webroot src="http://www.mydomain.com/[eWebEditProPath]/upload"/>
    </transport>
    <command name="cmdmfumedia" style="icon">
        <image key="Picture"/>
        <caption localeRef="btnTxtPic">Picture</caption>
        <toolTipText localeRef="btnPic">Insert Picture</toolTipText>
    </command>
    <validext>gif,jpg,png,jpeg,jif</validext>
</mediafiles>
```

MediaFile Object

The MediaFile object is the interface to a selected file's internet properties. Effective use of the MediaFile object is the key to manipulating the upload mechanism. The object contains information on uploading, referencing, and displaying the file.

To retrieve the interface to the object, using the `MediaFile` method.

```
var objMedia = eWebEditPro[sEditorName].MediaFile();
or
```

```
var objMedia = eWebEditPro.instances[sEditorName].editor.MediaFile();
```

Use these MediaFile object methods to affect upload information.

```
getProperty(PropertyName as string, vData as Variant)
getPropertyString(PropertyName as string) as Variant
```

Use these properties to affect the upload mechanism, location, and reference.

```
TransferMethod as String - Method of Upload
DefDestinationDir as String - Upload path location
(TransferRoot as String - alias for DefDestinationDir)
WebRoot as String - Path reference from a Web page
```

WebRoot only inherits the value of DefDestinationDir when it is not assigned a value anywhere, including in the XML configuration data. As a result, ASP, Cold Fusion, and other scripting that use the same domain and directory structure for transfers and referencing can work with the single DefDestinationDir property.

Modifying the Upload Location

XML Configuration Data

Any customization should begin with the XML configuration data, since all default settings are defined there. In this case, the mediafiles section is the focus. The following example explains how to address these issues.

- Since the example changes the upload mechanism programmatically, it defaults to the internal FTP setting.
- The default upload destination directory is defined in the configuration file.
- Since the Web reference path is the same as the destination, the example does not include a webroot element. As a result, the WebRoot property inherits the value defined in the xferdir element and any value assigned to the DefDestinationDir property.

Here is the mediafiles section of the configuration file for this example.

```
<mediafiles>
    <transport type="FTP">
        <domain></domain> <!-- empty means use current -->
        <xferdir src="[eWebEditProPath]/upload"/>
        <resolvemethod value="local"/>
    </transport>
    <command name="cmdmfumedia" style="icon">
        <image key="Picture"/>
        <caption localeRef="btnTxtPic">Picture</caption>
        <toolTipText localeRef="btnPic">Insert Picture</toolTipText>
    </command>
```

```
<validext>gif,jpg,png,jpeg,jif</validext>  
</mediafiles>
```

HTML Page

Here is sample HTML source page to start with. It is a simple page that contains the VisualFormat editor, a list selection item, and a text item to display the current upload path. The path manipulation functionality is added later.

```
<html>  
<html>  
<head>  
    <title>User Select Upload Location</title>  
    <script language="JavaScript1.2" src="ewebeditpro.js"></script>  
</head>  
<body>  
    <form method="post" name="selectpath">  
        <h1>User Select Upload Location</h1>  
        <script language="JavaScript1.2">  
            var g_strUserEditorName = "MyContent1";  
            document.write('<input type=hidden name=' + g_strUserEditorName +  
                '" value="This is initial content.">');  
            if (typeof eWebEditPro == "object")  
            {  
                eWebEditPro.create(g_strUserEditorName, "100%", 400);  
            }  
        </script>  
    </form>  
    <p><b>Current Upload Location:</b><br>  
    <select name="LocationSel" title="Current Upload Location"  
        OnChange="UseSelectedLocation(LocationSel.selectedIndex)">  
        <option>Configured Location<option>Work Location  
        <option>Publish Location</select><b>&nbsp;:&nbsp;</b>  
    <input name="CurUploadLocation" style="text" value="Upload Destination"  
        maxlength="256" size="80"></p>  
</body>  
</html>
```

Notice a call to the UseSelectedLocation function -- this is explained later.

Initialization

When the page loads, an external upload mechanism is assigned. The code also reflects to the user the current upload path from the MediaFile object.

```
<script language="JavaScript1.2">
var g_strConfiguredPath = "";
function eWebEditProReady(sEditorName)
{
    var objMedia = eWebEditPro[sEditorName].MediaFile();
    objMedia.setProperty("TransferMethod",
        "samples/asp/database/mediamanager.asp");
    g_strConfiguredPath =
        objMedia.getPropertyString("DefDestinationDir");
    document.selectpath.CurUploadLocation.value = g_strConfiguredPath;
}
</script>
```

The current path (the default path defined in the configuration file) is stored for later use. In our example, this allows the user to re-select it.

Note on the Missing eWebEditProReady

To the core Javascript files installed with VisualFormat, eWebEditProReady is a reserved function name. This function does not exist in the core Javascript files. Instead, a developer must define the function either in his/her own Javascript files or in the HTML file.

The function can be defined in any file brought in by the page. In the example below, this function is defined in the HTML file.

When the editor control sends the "ready" notification, the core Javascript checks to see if eWebEditProReady is defined. If it is, the core Javascript calls the function to notify the scripts that the editor is ready.

User Selection – Changing the Upload Location

In this example, the user can select one of three image upload locations. Here is the JavaScript to handle the selection:

```
function AssignUploadLocation(sEditorName, sLocation)
{
    var objMedia = eWebEditPro[sEditorName].MediaFile();
    objMedia.setProperty("DefDestinationDir", sLocation);
    document.selectpath.CurUploadLocation.value =
        objMedia.getPropertyString("DefDestinationDir");
```

```
        }
        function UseSelectedLocation(iIndex)
        {
            var UploadLoc = new Array(g_strConfiguredPath,
                eWebEditProPath + "samples/common/database",
                "/publish/images");
            AssignUploadLocation(g_strUserEditorName,
                UploadLoc[iIndex]);
        }
    }
```

The `UserSelectedLocation` function uses the index passed down from the `LocationSelection` item to specify the user's selection. `UserSelectedLocation` then sends the path associated with that selection to the `AssignUploadLocation` function.

The important function to examine is `AssignUploadLocation`, which illustrates how to set the destination directory for any uploads. This function receives the location and sets it into the `MediaFile` object.

Full Example

Here is the full HTML page that shows how to change the upload location.

In the example, the two important script functions are `eWebEditProReady` and `AssignUploadLocation`. These functions show how to access and modify the upload method and location.

```

<html>
<head>
    <title>User Select Upload Location</title>
    <script language="JavaScript1.2" src="ewebeditpro.js"></script>
</head>
<body>
<script language="JavaScript1.2">
var g_strConfiguredPath = "";
function eWebEditProReady(sEditorName)
{
    var objMedia = eWebEditPro[sEditorName].MediaFile();
    objMedia.setProperty("TransferMethod",
        "samples/asp/database/mediamanager.asp");
    g_strConfiguredPath =
        objMedia.getPropertyString("DefDestinationDir");
    document.selectpath.CurUploadLocation.value = g_strConfiguredPath;
}
function AssignUploadLocation(sEditorName, sLocation)
{
    var objMedia = eWebEditPro[sEditorName].MediaFile();
    objMedia.setProperty("DefDestinationDir", sLocation);
    document.selectpath.CurUploadLocation.value =
        objMedia.getPropertyString("DefDestinationDir");
}
function UseSelectedLocation(iIndex)
{
    var UploadLoc = new Array(g_strConfiguredPath,
        eWebEditProPath + "samples/common/database",
        "/publish/images");
    AssignUploadLocation(g_strUserEditorName,
        UploadLoc[iIndex]);
}
</script>
<form method="post" name="selectpath">
<h1>User Select Upload Location</h1>
<script language="JavaScript1.2">
var g_strUserEditorName = "MyContent1";
document.write('<input type=hidden name=' + g_strUserEditorName +
    ' value="This is initial content.">');
if (typeof eWebEditPro == "object")
{
    eWebEditPro.create(g_strUserEditorName, "100%", 400);
}
</script>
<p><b>Current Upload Location:</b><br>
<select name="LocationSel" title="Current Upload Location"
    OnChange="UseSelectedLocation(LocationSel.selectedIndex)">
    <option>Configured Location<option>Work Location
    <option>Publish Location</select><b>&nbsp;&nbsp;</b>
<input name="CurUploadLocation" style="text" value="Upload Destination"
    maxlength="256" size="80"></p>
</form>
</body>
</html>

```

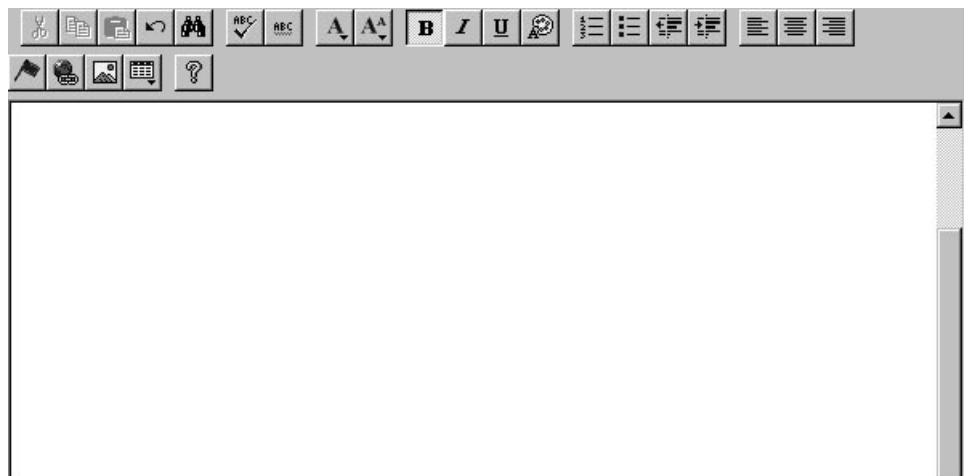
Integrating VisualFormat Using JavaScript

Formats for Placing the Editor on the Page

You can place the editor as a

- box whose width and height you specify, or
- button that, when pressed, displays the editor in the text field.

This illustration depicts the editor appearing as a box.



This illustration depicts the editor as a button. When the user clicks the button, the editor appears in the window.



Creating Your Own Page

If you want to place VisualFormat on an HTML page, the page needs to include these actions.

1. Create an HTML page with header and body tags.
2. Include the VisualFormat javascript file.
3. Set up a form.
4. Modify the parameters (optional).
5. Create an input area.
6. Invoke the editor.

The rest of this section explains how to complete these tasks.

Create an HTML Page with Header and Body Tags

Set up a typical HTML page.

```
<HTML>
<head>
<title>VisualFormat</title>
</head>
<body>
</body>
</HTML>
```

Include the VisualFormat JavaScript File

Your page must include a reference to the ewebeditpro.js file. Place the src reference within the head tags.

```
<script language="JavaScript1.2" src="/ewebeditpro2/ewebeditpro.js"></script>
```

Enter a Form Element

Within the body tags, enter a set of form tags. Assign the method attribute to the form tags, and post as the method's value.

```
<form method="post">
place the editor here
</form>
```

If a form element's method is not set to post, an error message appears below the editor.

Changing Parameter Values

If you want to change parameters that affect all instances of the editor, edit the ewebeditprodefaults.js file using a standard text editor (see "The ewebeditprodefaults File" on page 76).

To change the parameters only for the instance of VisualFormat that you are placing on the HTML page, enter the following code.

```
<script language="JavaScript">
eWebEditPro.parameters.parameter= "value";
</script>
```

For example, the following code displays the "about" button on the toolbar.

```
<script language="JavaScript">
eWebEditPro.parameters.hideAboutButton="False";
</script>
```

If you are placing more than one editor on a page, and you want the parameters for each editor to be different, begin the parameter code with `eWebEditPro.parameters.reset()`. This line restores the parameters to the default values set in ewebeditprodefaults.js.

Inserting the Editor as a Box

Create a Content Field

Within the form tags, enter a hidden field or text input type tag, or a or textarea tag. This example uses a hidden field.

```
<input type=hidden name="MyContent1" value="This is initial content.">
```

See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.

NOTE

Please read "Encoding Characters in the Value Attribute" on page 293 for important details about the value attribute.

Declaring a Content Field after Creating the Editor

If the content field (typically a hidden field) appears after the `eWebEditPro.create` code, the following error message appears below the editor.

Content field must be declared prior to creating the editor.

If you must declare the control field after creating the editor, delete the error message before entering the create command.

```
eWebEditProMessages.elementNotFoundMessage="";
eWebEditPro.create(...);
```

The content will still be loaded, but it cannot check to see if the content field exists.

Creating the Editor

Within the input area, place the editor using the following JavaScript.

```
<script language="JavaScript1.2">
<!--
eWebEditPro.create("field name", width, height);
//-->
</script>
```

Argument	Description
field name	Enter the name of the field within quotes (" "). The field name must match the value of the name attribute in the input type tag.
width, height	Enter the width and height of the editor in percent or pixels. If a percent, enclose the value in quotes (" ") and follow it with a percent sign (%), for example "50%". If pixels, quotes are optional, for example, 500.

For example

```
<script language="JavaScript1.2">
<!--
eWebEditPro.create("MyContent1", 700,150);
//-->
</script>
```

The content is automatically loaded into the editor and automatically saved when the form is submitted.

NOTE

If you submit by calling the form.submit method, you must manually save the content by calling eWebEditPro.save() just prior to calling form.submit. To learn how content is loaded and saved, see "Loading the Content" on page 295.

To access the ActiveX control via JavaScript, once an instance of the editor is created, use the VisualFormat JavaScript object.

```
eWebEditPro.theeditorname
```

The ActiveX control should not be accessed until after the VisualFormat onready event fires.

For example

```
eWebEditPro.MyContent1.pasteHTML( "<HR>" ); // insert horz rule
```

The pasteHTML method inserts HTML content into the editor. For more information on the pasteHTML and other methods, properties and events of the VisualFormat ActiveX control, see “ActiveX Control” on page 115.

Inserting the Editor as a Button

Entering a Field

Within the form tags, enter a hidden field or text input type tag, or a textarea tag. This example uses a textarea declaration.

```
<textarea name="tal" cols=80 rows=5>Click 'Edit' to edit this content with the  
eWebEditPro editor.  
</textarea>
```

NOTE

If you decide to use hidden field or a text field, read “Encoding Characters in the Value Attribute” on page 293 for important details about the value attribute.

Entering the Button

To add the button to the page, enter a line with the following elements.

```
<script language="JavaScript1.2">  
  <!--  
  eWebEditPro.createButton( "button name" , "textarea name" );  
  //-->  
</script>
```

Argument	Description
button name	Enter a button name within quotes (" "). This is the field name of the button that by default is <input type=button...>. To change the button type, see “Customizing the Popup Button” on page 43.
textarea name	Enter the name of the textarea within quotes (" "). The field name must match the field named in the textarea declaration.

For example

```
<script language="JavaScript1.2">
<!--
eWebEditPro.createButton("btnEdit", "tal");
//-->
</script>
```

NOTE

To edit the button text, open the ewebeditpromessages.js file using a standard text editor. Within that file, edit the text within quotes that follows **popupButtonCaption:**.

Encoding Characters in the Value Attribute

Initial content for the editor is typically stored in the value attribute of a hidden field. For example,

```
<input type="hidden" name="MyContent1" value="This is the initial content.">
```

If the content includes a quote ("), greater or less than character (<>), or an ampersand (&), the browser prematurely terminates the display of the content. For example, the input declaration

```
<input type="hidden" name="MyContent1" value="Characters that need to be encoded: " & <tag>">
would display the following text in the editor
```

Characters that need to be encoded:

This problem occurs because the browser cannot distinguish between one of these characters and the delimiters of the value attribute.

Also, if you use single quotes to delimit the value attribute, which is not recommended, you need to encode all single-quote characters.

To solve this problem, you must encode these characters when storing them in a hidden field. You would insert the character's *entity* or *character reference* in place of the actual character in the value field.

The following table lists the characters and corresponding entity and character reference values.

Character	Entity	Character Reference	Comments
&	&	&	Must be encoded first.
>	<	<	
<	>	>	
"	"	"	Value attribute must be quoted with ", not '.

NOTE The order in which characters are encoded is important. The ampersand (&) must be encoded before you encode the other characters.

How the Server Converts Characters

Your web application server must convert these characters. For example, ASP offers the Server.HTMLEncode function. If your environment does not provide such a function, you need to write it. It is straightforward and requires the use of a string substitution function. The pseudo code to encode these characters appears below.

```
String strContent
```

```
strContent = ReplaceString(strContent, "&", "&amp;")  
strContent = ReplaceString(strContent, "<", "&lt;")  
strContent = ReplaceString(strContent, ">", "&gt;")  
strContent = ReplaceString(strContent, "", "&quot;")
```

Encoding the Single Quote

We recommend surrounding the value attribute with double quotes, but if you decide to use single quotes, you must encode the single quote character (also known as an apostrophe).

Character	Entity	Character Reference	Comments
'	' (but see comments)	'	&apos; is for XML parsers but may not be supported by an HTML browser. Therefore, the character reference (') is preferred, because HTML browsers <i>and</i> XML parsers support it.

Content Stored in a Textarea Field

When stored in a TEXTAREA field, the greater/less than characters (<>) do not need to be encoded, because TEXTAREA does not use a value attribute. The double-quote ("), single-quote ('') and ampersand (&) characters should be encoded in a TEXTAREA field, although most browsers will accept them without encoding.

Loading the Content

Content is loaded into the editor during the page's onload event, which invokes the eWebEditPro.load method. The method copies content from the hidden field (or other HTML element) to the editor.

To prevent loading, set window.eWebEditProLoadHandled to **true**.

```
window.eWebEditProLoadHandled=true;
```

Detecting the Load Method

To detect when the load method is being invoked, two VisualFormat events are fired at this time.

- onbeforeload
- onload

If the onbeforeload event handler returns false, it terminates the load method and the onload event.

For example,

```
eWebEditPro.onbeforeload="return confirm('Do you want to load?');  
eWebEditPro.onload="alert('Done loading.');" ;  
eWebEditPro.create(...);
```

NOTE

In Netscape, the alert may not function during onload events.

Manually Loading Content into the Editor

To let the user manually load content, use this syntax.

```
window.eWebEditProLoadHandled=true  
eWebEditPro.create(...)  
</script>  
<input type=button value="load" onclick="eWebEditPro.load()">
```

The `eWebEditPro.load` method loads all instances of the editor on the page. To load just one, you can use

`eWebEditPro.instances[n].load()`. Or, you can pass the content using `eWebEditPro.instances[n].load(strContent)`.

If you use `eWebEditPro.instances[n].load()`, the `n` within square brackets is either the name of the editor used when creating it (for example, `eWebEditPro.create("EditorName", ...)`) or an index number (0, 1, 2, etc., where 0 is the first editor created).

See Also: “Appendix A: Naming the VisualFormat Editor” on page 300.

For example

```
eWebEditPro.create("Summary", 700, 200);  
eWebEditPro.create("Teaser", 700, 300);  
eWebEditPro.create("Desc", 700, 400);  
...  
eWebEditPro.instances["Desc"].load();  
or  
eWebEditPro.instances[2].load();
```

Saving the Content

Content is saved (that is, copied to the hidden field) during the form’s `onsubmit` event, which invokes the `eWebEditPro.save` method. This method copies the content from the editor to the hidden field (or other HTML element). In Internet Explorer, the content is also saved when the page is unloaded.

NOTE

The `eWebEditPro.save` method saves content to a temporary cache in the browser. The content is saved permanently when the form is submitted and its fields are posted to the server.

To prevent saving, set the form's eWebEditProSubmitHandled method to **true**.

```
document.yourformname.eWebEditProSubmitHandled=true;
```

Detecting when the Save Method is Invoked

To detect that the save method is being invoked, two VisualFormat events are fired at this time.

- onbeforesave
- onsave

Terminating the Save Method

Returning false in the onbeforesave event handler terminates the .save method and the onsave event.

For example

```
eWebEditPro.onbeforesave="return confirm('Do you want to save?');  
eWebEditPro.onsave="alert('Done saving')";
```

Saving Content Manually

To manually save content, use this code.

```
function mysubmit(){  
    eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_NOSAVE;  
    eWebEditPro.save();  
    document.myform.submit();
```

Closing a Window without Saving Content

To close a window (that is, cancel) without saving the content to the hidden field, use this code.

```
eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_NOSAVE;  
self.close();
```

Prevent Detecting the onsubmit Event

To prevent automatic saving of the content to the hidden field when a submit button is pressed, use this code.

This code must appear *prior* to creating the editor on the page.

```
document.myform.eWebEditProSubmitHandled=true;
eWebEditPro.create(...);
```

Prevent Detecting the onbeforeunload/onunload Event

To prevent automatically saving the content to the hidden field when the web page is unloaded, use this code.

This code must appear *prior* to creating the editor on the page.

```
window.eWebEditProUnloadHandled=true;
eWebEditPro.create(...);
```

Saving from One Instance of the Editor

The eWebEditPro.save method saves all instances of the editor on the page. To save just one, use

eWebEditPro.instances[n].save(). The **n** within square brackets is either the name of the editor used when creating it (for example, eWebEditPro.create("EditorName", ...)) or an index number (0, 1, 2, etc., where 0 is the first editor created).

See Also: "Appendix A: Naming the VisualFormat Editor" on page 300.

For example

```
eWebEditPro.create("Summary", 700, 200);
eWebEditPro.create("Teaser", 700, 300);
eWebEditPro.create("Desc", 700, 400);
...
eWebEditPro.instances["Desc"].save();
```

or

```
eWebEditPro.instances[2].save();
```

Alternatively, you can retrieve content by passing an object to the save method. To do this, set the object's value property to receive the content.

For example

```
var objContent = new Object();
objContent.value="";
eWebEditPro.save(objContent);
```

.objContent.value now stores the content.

Detecting When the Popup Editor is Activated

Similarly, when using a popup editor with VisualFormat.createButton(), there are two VisualFormat events that fire when the button is pressed and when the popup window is closed.

- onbeforeedit
- onedit

For example

```
eWebEditPro.onbeforeedit="return confirmed('Do you want to edit?');  
eWebEditPro.onedit="alert('Done editing');  
eWebEditPro.createButton(...);
```

Appendix A: Naming the VisualFormat Editor

When you are naming the VisualFormat editor, the name must be a valid JavaScript identifier. As a result, the name must follow these guidelines.

- It consists of only ASCII letters and digits, underscores (_) and dollar signs (\$).
- The first character cannot be a digit.
- Spaces are not permitted.

Appendix B: Error Messages

Error Message	Cause	How to Resolve	Audience Where Message is Defined
A license is required for host:	The license key does not match the host name.	The developer must specify a valid license key.	End user/Developer locale0000.xml
Click OK to preserve changes when moving to another page. Click Cancel to discard changes.	Unloading a web page with the editor prompts to cache the content in the content field. (Only with IE.)	The end user can click OK or Cancel. The developer can change the value of actionOnUnload.	End user ewebeditpromessages.js
Content is too large to save. Please reduce the size and try again.	The size of the HTML content in the editor is larger than the amount specified in maxContentSize.	The end user can reduce the content. The developer has several options. For more information, see http://www.ektron.com///support/ewebeditprokb.cfm?doc_id=1326 and http://www.ektron.com///support/ewebeditprokb.cfm?doc_id=1204	End user ewebeditpromessages.js
Error uploading the selected file.
The uploading of files may not be allowed at this location. Please verify that the connection settings and server permissions are correct.	The server is not allowing the uploading of files. The wrong server may have been specified in the login information. The login account may not have upload permissions.	1. Verify that the server is the correct server used for uploading files. 2. Contact the site administrator to ensure that the login account has upload permissions.	End user locale0000.xml
VisualFormat cannot clean the document until these errors are fixed.	The HTML content is corrupt and could not be adequately cleaned.	Manually fix the corruption and try again.	End user locale0000.xml

Error Message	Cause	How to Resolve	Audience Where Message is Defined
VisualFormat is not installed. Click to install VisualFormat.	The editor has not been installed yet.	Install the editor using the client installation program.	End user ewebeditpromessages.js
Internet Explorer 4.0 or later is required.	IE 3.x or older is being used.	IE 4.01 or later is required.	End user locale0000.xml
Invalid License	<p>The license key is invalid. Likely reasons include</p> <ol style="list-style-type: none"> 1. The domain name in the URL does not match one of the license keys. For example, http://www.ektron.com matches www.ektron.com?123456, but http://www.ektron.com does not match 123.045.067.089?123456. 2. The license key is expired 3. No license key is specified 4. The license key is for another product or version 5. The license key is corrupt 	<ol style="list-style-type: none"> 1. The domain name in the browser's address or location bar must match one of the license keys. For example, http://www.ektron.com matches www.ektron.com?123456, but http://www.ektron.com does not match 123.045.067.089?123456. Use either the name specified in the license key or purchase another license key for the domain name. 2. Purchase a license key. 3. Specify a valid license key. See also the Knowledge Base article "Error Message: Invalid license with no license keys in box" at http://www.ektron.com///support/ewebeditprokb.cfm?doc_id=936. 4. Purchase a license key for this product or version. License keys for VisualFormat 1.8 are not valid for VisualFormat 2.0. 5. Specify the entire license key and ensure all the numbers are correct. For example, www.ektron.com?123456-20, not just 123456-20. 	End user. Appears in the About box, which pops up if a valid license is not specified. locale0000.xml

Error Message	Cause	How to Resolve	Audience Where Message is Defined
License is expired for date:	The license key is expired.	The developer must specify a valid license key.	End user/Developer locale0000.xml
Sorry, the connection could not be established. Please verify that the login and connection information are correct.	The connection to the server could not be established. This could be caused by incorrect server address or login information.	Verify that the server domain, login name and password are correct. If they are, contact the administrator of the remote site to verify that the login information is correct.	End user locale0000.xml
The editor was not able to create the DHTML Editor. Please run the client installation or contact your system administrator.	The editor was not able to create a critical component because one or more required files are missing or corrupt.	Try installing using the client installation program.	End user locale0000.xml
The editor was not able to create the HTML Source View Editor. Please run the client installation or contact your system administrator.	The editor was not able to create a critical component because one or more required files are missing or corrupt.	Try installing using the client installation program.	End user locale0000.xml
The editor was not able to create the Toolbar. Please run the client installation or contact your system administrator.	The editor was not able to create a critical component because one or more required files are missing or corrupt.	Try installing using the client installation program.	End user locale0000.xml
The form method must be set to "post". For example, <form method="post">. The submit will fail using "get".	The form's method is not set to post.	The developer must set the method to post.	Developer ewebeditpromessages.js

Error Message	Cause	How to Resolve	Audience Where Message is Defined
The page content is still initializing. Please wait...	<p>The web page with the editor is still loading and initializing when the end user pressed a toolbar button.</p> <p>This message is typically only seen when using Netscape.</p>	Wait a few seconds and try again.	End user locale0000.xml
The selected file is too large to allow an upload. The maximum size is	The size of the target upload file exceeds the upload limits defined by the site administrator.	<ol style="list-style-type: none"> 1. Select a file that is less than the maximum allowed size. 2. Ask the site administrator to increase the size limit specified in the XML configuration data. 	End user locale0000.xml
There is excessive HTML code that may prevent you from changing text format.	Prompt to clean Office/Word 2000 content	It is recommended that you clean the content.	End user locale0000.xml
There was an error in the dialog. The client installation for the editor may need to be run to correct the issue.	The editor was not able to open a dialog window because one of the required files is missing or out of date.	Try installing using the client installation program.	End user locale0000.xml
Unable to check spelling. Microsoft Word 97 or later is required.	Spell checking is not supported because Word 97 or later is not installed or cannot be accessed.	Install Microsoft Word 97 or later.	End user locale0000.xml

Error Message	Cause	How to Resolve	Audience Where Message is Defined
<p>Unable to find content field (typically a hidden field) within a form. Please check the following:</p> <ul style="list-style-type: none"> ● Form tag is required ● Content field is required and must match the name specified when creating the editor ● Content field must be declared prior to creating the editorName specified: 	<p>The editor could not find the content field.</p>	<p>Please check the following:</p> <ul style="list-style-type: none"> ● Form tag is required ● Content field is required and must match the name specified when creating the editor ● Content field must be declared prior to creating the editor 	Developer ewebeditpromessages.js
<p>Unable to run in container:</p>	<p>The editor is being used in an application other than a web browser.</p>	<p>The editor can only be used in a browser.</p>	Developer locale0000.xml
<p>Unable to save. Continue and lose content?</p>	<p>The editor is unable to save the content in the content field, typically because the window with the content field was closed.</p>	<p>The end user can copy the content to the clipboard to preserve it.</p>	End user ewebeditpromessages.js

Index

A

about window, displaying 172
ActiveX
 control properties
 BaseUrl 116
 CharSet 118
 Config 118
 hideAboutButton 118
 License 118
 Locale 118
 srcPath 119
 StyleSheet 119
 Title 119
 version 120
events
 ondblclickelement 131
methods 121
 ExecCommand 121
 for style sheets 124
 getBodyHTML 121
 getBodyText 121
 getDocument 121
 getHeadHTML 121
 getProperty 122
 getPropertyBoolean 122
 getPropertyInteger 122
 getPropertyString 122
 getSelectedHTML 122
 getSelectedText 122
 pasteHTML 122
 pasteText 122
 setBodyHTML 122
 setDocument 123
 setHeadHTML 123
 setProperty 124
style sheets
 addInlineStyle 124
 addLinkedStyle Sheet 125
 BodyStyle 126
 ClearStylesFromTags 128
 disableAllStyleSheets 125
 disableStyleSheet 125
 GetActiveStyleSheetTitles
 129
 PopulateTagsWithStyles 127
 ShowActiveStylesDetails 130
ASP file upload 252
attribute types, configuration file 147
attributes
 removing from file globally 201

B

background color, setting for editor 117
bar element, configuration file 148
boolean attribute type 147
button element, configuration file 149
buttons
 adding separator bar between 20
 adding space between 19
 adding to menu 17
 caption text
 aligning 22
 displaying 21
 images
 changing 20
 creating custom 184
 rearranging on menu 19
 removing from menu 18
 translating to foreign language
 23

C

caption
 menu,editing 16
caption element,configuration file 150
carriage return,processing when
 content is saved 199
cascading style sheets, see style
 sheets
characters
 encoding in the Value attribute
 293
 special and extended, see
 encoding special characters
charencode Attribute 210
 binary 211
 charref 212
 entityname 212
 latin 213
 special 213
 tips for choosing 213
 UTF-8 211
charset, specifying for a page 118
class
 style
 apply to selected text
 224
 suppressing from drop-
 down list 227
 translating to foreign lan-
 guage 226
 types 225
class tags, removing from Microsoft
 WORD 2000 content 223

cleanHTML 197
client installation
 file
 directory path to 107
pages
 customizing 84
 deleting 85
client installation file
 directory path to 76
cmdbold 172
cmdbookmark 172
ColdFusion
 file upload 254
colors
 editing list on toolbar 173, 175
 restricting user access 173, 175
command element,configuration file
 152
command object interface
 method
 AddItem 62
 Clear 63
 FirstCommand 65
 getProperty 65
 getProperty String 66
 getPropertyBoolean 67
 getPropertyInteger 66
 IsValid 67
 ListCommandName 68
 NextCommand 68
 SetProperty 69
property
 CmdCaption 63
 CmdData 63
 CmdGray 64
 CmdIndex 64
 CmdName 64
 CmdStyle 64
 CmdText 64
 CmdToggledOn 64
 CmdToolTipText 64
 CmdType 64
 CmdVisible 65
 MaxListboxWidth 68
commands 171
 custom 25, 178
 executing via JavaScript 24
guidelines for using 172
modifying by scripting 47
remove from context menu 41
standard
 cmdabout 172
 cmdbold 172
 cmdbookmark 172

cmdbullet 172
cmdcenter 172
cmdclean 172
cmdcopy 172
cmdcut 172
cmddelete 172
cmdfind 172
cmdfontcolor 173
cmdfontcolorvalue 173,
 175
cmdfontname 173
cmdfontsize 173
cmdheaderlevel 173
cmdhelp 173
cmdhr 173
cmdhyperlink 173
cmdimage 174
cmdindentleft 173
cmdindentright 173
cmditalic 173
cmdleft 173
cmdmfunsert 174
cmdnumbered 174
cmdpaste 174
cmdpicture 174
cmdredo 174
cmdright 174
cmdselstyle 174
cmdstrike 174
cmdsub 174
cmdsupserscript 174
cmdtable 174
cmdunderline 174
cmdundo 174
cmdunlink 174

config element, configuration file 155
config.xml file see configuration file

configuration file
 allowCustomize 137
 allowcustomize 137
 attribute types 147
 button 149
 changing location 136
 clean 197
 command 171
 config 146
 customizing 144
 editHTML 196
 editing 134
 elements
 hierarchy 145
 external 193
 features element 146
 interface 146
 interface element 146
 managing 132, 134
 mediafiles feature 256

 defsource element 263
 domain element 260
 maxsizek element 257
 mediafiles element 256
 password element 259
 port element 264
 proxyserver element 260
 resolvemethod element
 264
 transport element 258
 username element 259
 validext 256
 webroot element 262
 xferdir element 261
 menu 164
 overview 141
 spellcheck 204
 standard element 168
 standard elements 144
 bar 148
 button 149
 caption 150
 command 152
 config 155
 features 156
 image 157
 interface 159
 listchoice 161
 menu 164
 popup 166
 selection 166
 space 167
 tooltiptext 169
 table 186
 using to customize toolbar 12, 132
 viewas 195
content
 loading into editor 295
 saving in editor 296
context menu
 displaying 160
 suppressing 160
context menu, see menus, context
custom commands 25, 178
custom features
 creating 193
custom Javascript function, creating 25

D

 default style sheet 220
 dialog boxes
 translating to another language 71
 directory path to eWebEditPro 75

E
editHTML feature, configuration file 196
editor
 changing dynamically with
 JavaScript, client side 38
 changing dynamically with
 JavaScript, server side 37
 inserting as a box 290
 inserting as a button 292
 loading content 295
 naming guidelines 300
 placing on a web page 288
 popup, detecting when activated
 299
 saving content 296
encoding
 characters in the Value attribute
 293
special characters 207
 configuring VisualFormat
 209
 displaying Asian languages 209
 preventing for certain
 characters 199
end tag, removing 200
estimating size of content 91
event handler functions 96
 eWebEditProDblClickElement 98
 eWebEditProDblClickHyperlink 99
 eWebEditProDblClickImage 99
 eWebEditProDblClickTable 99
 eWebEditProExecCommand 97
 eWebEditProMediaSelection 98
 eWebEditProReady 98
ewebeditpro.js file
 eWebEditProPath 75
 including 289
 LicenseKeys 75
eWebEditProDblClickElement 98
eWebEditProDblClickHyperlink 99
eWebEditProDblClickImage 99
eWebEditProDblClickTable 99
ewebeditprodefaults.js file
 clientInstall 76, 107
 embedattributes 79, 107
 maxContentSize 79
 objectattributes 79, 108
 onblur 108
 ondblclickelement 79, 108
 onExecCommandDeferred 109
 onfocus 109
 path 76, 109
 popupButtonTagEnd 79, 110
 popupButtonTagStart 79, 110
 popupQuery 78, 111, 112
 popupURL 76, 112

<p>popupWindowFeatures 76, 77, 113 popupWindowName 76, 113 textareaAttributes 113 ewebeditproevents.js file onDbClickElementHandler 83 onDbClickHyperlinkHandler 83 eWebEditProExecCommand 97 eWebEditProMediaSelection 98 ewebeditpromessages.js file clientInstallMessage 81 confirmAway 81 doneLoading 80 doneSaving 80 elementNotFoundMessage 81, 82 errorLoading 80 installPrompt 80 invalidFormMethodMessage 82 loading 80 popupButtonCaption 80 querySave 81 saveFailed 81 saving 80 sizeExceeded 81 waitingToLoad 80 eWebEditProReady 98 extended characters, see encoding special characters external features adding 193</p>	<p>form element, entering on a web page 289 FTP file upload 250 image selection example 244 function, custom, creating 25 functions, event handler (see event handler functions)</p>	<p>FWPort 267 FWUse 268 FWUsePassV 268 HandledInternally 268 HorizontalSpacing 268 LoginName 268 MaxFileSizeK 268 NeedConnection 268 Port 268 ProxyServer 268 RemotePathFileName 268 ResolvePath 269 ShowHeight 269 ShowWidth 269 SrcFileName 269 TransferMethod 269 TransferRoot 269 Use PassV 269 ValidConnection 269 ValidExtensions 269 VerticalExtensions 269 WebPathName 270 WebRoot 270</p>
<p>F</p> <p>features element, configuration file 156 file upload information object properties FW Password 267 FWProxyServer 268 ImageHeight 268 ImageWidth 268 IsLocal 268 Password 268 ProxyServer 268 ResolveMethod 269 TransferRoot 269</p> <p>font name, specifying in the configuration file 187</p> <p>font size changing list of 34 specifying in the configuration file 189</p> <p>fonts assigning color 173 changing list of 34 default, specifying 34 specifying in the configuration file 187</p>	<p>image element, configuration file 157 image file dynamically selecting upload location 281</p> <p>image selection database samples 247 examples of implementing 236</p> <p>FTP example 244 requirements 245 preventing user from upload 258 workflow 234</p> <p>image selection object methods 266 properties 266 accessing programmatical- ly 271 alignment 266 AllowSubDirectories 267 AllowUpload 267 BaseUrl 267 BorderSize 267 ConfirmBeforeUpload 267 DefDestinationDir 267 DefSourceDir 267 Domain 267 FileSize 267 FileTitle 267 FileType 267 FWLoginName 267</p>	<p>image upload implementing 250 see also image selection</p> <p>images repository, setting up 278</p> <p>including the VisualFormat Javascript file 289</p> <p>installation pages, client, customizing 84</p> <p>instance object 102</p> <p>events 105 onerror event 105</p> <p>methods 104 load 104 save 104</p> <p>properties 102 editor 102 elemName 102 formName 103 height 103 html 103 id 103 maxContentSize 103 name 102 receivedEvent 102 status 103 width 103</p> <p>integer attribute type 147</p> <p>integrating VisualFormat using JavaScript 288</p> <p>interface element, configuration file 159</p> <p>interface, user</p>

J

JavaScript
 custom function
 creating 25
Javascript
 using to execute commands 24
Javascript files, customizing 75
JavaScript object 86
 events
 onbeforeedit 94
 onbeforeload 94
 onbeforesave 94
 oncreate 93
 oncreatebutton 94
 onedit 94
 onerror 96
 onload 95
 onsave 95
 methods
 autoInstallExpected 90
 create 90
 createButton 91
 edit 91
 estimateContentSize 91
 load 93
 refreshStatus 93
 save 93
properties 87
 {editor name} 87
 actionOnUnload 87
 installPopup 87
instances 87
isAutoInstallSupported 88
isInstalled 88
isSupported 88
parameters 88
status 88
 EWEP_STATUS_FATALERR
 OR 88
 EWEP_STATUS_LOADED 88
 EWEP_STATUS_LOADING 88
 EWEP_STATUS_NOTINSTAL
 LED 88
 EWEP_STATUS_NOTLOADE
 D 88
 EWEP_STATUS_NOTSUPPO
 RTED 88
 EWEP_STATUS_SAVED 88
 EWEP_STATUS_SAVING 88
 EWEP_STATUS_SIZEEXCEE
 DED 88
 EWEP_STATUS_UNABLETO
 SAVE 88
upgradeNeeded 89
versionInstalled 89

L

language
 foreign
 spell checking 74
 translating dialog boxes
 and menus 71
 selecting
 default 71
 dynamically 72
 VisualFormat screens and m
 enus, changing 70
license keys
 enter/edit 75
linefeed character, processing when
 content is saved 199
list item
 creating that generates no comm
 and 35
listchoice element, configuration file 161
localization file 70
 matching to regional settings of
 client 72
 specifying a language 72

M

maxContentSize 108
maxContentSize, defining 103
mediafile object
 using 282
mediafile object properties
 accessing programatically 271
 accessing with Netscape 272
 changing transfer method on the
 fly 275
modifying upload directory 276
setting external page parameters
 274
specifying image 275
using external scripts 273
menu element, configuration file 164
menus
 adding
 buttons 17
 separator bar between 20
 space between buttons 19
 to toolbar 15
aligning button caption text 22
caption, editing 16
changing image on buttons 20
context
 customizing 41
 removing commands 41
 suppressing 42
context, see context menu
creating 15
defining 12

disabling and enabling with
 JavaScript 39
displaying button caption text 21
modifying by scripting 47
placing on row with another menu
 15
popup
 creating 31
rearranging buttons 19
removing buttons 18
removing from toolbar 14, 15
right-click, see context menu
right-mouse click, see m
 enus, context
translating buttons to foreign
 language 23
translating to another language 71
wrapping to new toolbar row 16
menus interface
method
 CommandAdd 55
 CommandDelete 55, 56
 CommandItem 56
 HideAbout 56, 57
 HideAllMenus 57
 PopupMenu 57
 SeparatorBarAdd 58
 SeparatorSpaceAdd 59
 ShowAbout 59
 ShowAllMenus 60
 ToolbarAdd 60
 ToolbarModify 61
menus object interface 47
menus, user customization 137
messages
 translating to another language 72
Microsoft Office 2000 content, preparing
 for copying to VisualFormat 169
Microsoft Word
 content, cleaning 197
Microsoft Word 2000
 content
 removing class and style
 tags 223
minimum size needed to show clean
 HTML dialog box 200

O

object
 eWebEditPro 86
properties
 InstallPopupQuery 110
 InstallPopupwindowFea
 tures 109
onerror event, instance object 105

P

parameter values, changing for one editor 290
parameters object 106
 methods 114
 reset 114
 properties 106
 baseURL 116
 bodyStyle 117
 buttonTag 107
 charset 118
 cols 107
 config 118
 editorGetMethod 107
 hideaboutbutton 118
 installPopupurl 109
 InstallPopupwindowName 109
 maxContentSize 108
 rows 113
 title 119
path to ewebeditpro 75
popup button
 customizing 43
popup editor, detecting when activated 299
popup element, configuration file 166
popup menu
 creating 31
popup window
 buttons that launch 79, 110
 features 77, 113
 query 78, 111, 112
 specifying name 76, 113
 specifying web page of 76, 112
preservewordstyles attribute 223
publish, attribute of standard element 169
publishStyles attribute 223

Q

quick links
 editing list 232

R

redisplay toolbar command 102
right-click menu, see context menu

S

save method
 detecting when invoked 297
 terminating 297
saving content 296
 preventing
 when submit button is pressed 297
 when web page unloaded 298
saving editor content
 body only 107
 entire HTML document 107
selection element, configuration file 166
showonsize attribute, clean element 200
size of content, estimating 91
space element, configuration file 167
span tags, with font styles, converting to font tags 199
special characters
 commands 177
 see also encoding special characters
spell checker
 as you type 205
 enabling 204
 foreign language 74
 image that indicates misspelled word 205
 specifying number of replacement words 205
standard command, detecting when executed 29
standard element, configuration file 168
status bar
 translating to another language 72
string attribute type 147
Style Sheet, ActiveX control property 119
style sheets 219
 ActiveX methods 124
 applying 220
 applying a style class to selected text 224
 default 220
 publishStyle attribute's effect 222
 specifying
 for a page 222
 for one editor 222
 in config.xml 221
 suppressing style classes from drop-down menu list 227
 three levels 220
 translating style classes 226
style tags, removing from Microsoft WORD 2000 content 223

T

tables, enabling in the configuration file 186
tagelement element 202
tagonly element 202
tags
 HTML
 removing 202
 removing content between 202
 removing unnecessary 199
title, HTML page, setting using ActiveX control property 119
toolbarreset command 99
toolbars
 defining 12
 reacting to the creation of 99
 redisplaying 102
 user customization 137
tooltiptext element 169
translating VisualFormat to another language 70

U

unicode characters 209
upload
 destinations
 selecting dynamically 281
 directory
 selecting dynamically 281
user interface, defining 144
UTF-8 209, 211
UTF-8 encoding
 implementing a web site using 216

V

viewas feature 195
viewing HTML source code 195

W

web page
 creating 289
window
 closing without saving content 297

X

xhtml output, determining 169
xhtml output, specifying 169
xml files
 validating 12, 70, 132, 141